TURING

Programmers at Work

Interviews With 19 Programmers Who Shaped the Computer Industry

编程大师访谈录

[美] Susan Lammers 著

李琳骁 张菁 吴咏炜 译



人民邮电出版社
POSTS & TELECOM PRESS



Susan Lammers

现居华盛顿州西雅图市，是手机应用开发公司Flying Sofa的合伙人，也是风险投资资助的创业公司Headbone Interactive的董事长及创始人，这家公司专为互联网、电视及其他面向儿童的媒体开发交互式多媒体软件。Susan是多媒体领域的先锋，曾是微软公司早期的多媒体出版部联合出版人和总监，负责微软最早的交互式媒体项目，包括微软的电子百科全书Encarta（英卡塔）。

1990年，Lammers离开微软，加入一家创业公司，成为该公司的第三位员工。这家公司现名Corbis，已跻身世界最大的数字图像代理公司之列。Corbis由比尔·盖茨创立，旨在开发自然语言图像的大型数据库并进行交互式电视程序设计。

Lammers毕业于斯坦福大学，拥有英语文学士学位。

译者心语

李琳骁　从事嵌入式Linux内核/驱动开发，关注IT、开放源码和安防监控等领域。业余以技术翻译为乐，时而客串编辑，好为爱书挑错，渴求完美，却也常因“小”失大，不得读书要旨。翻译或参与翻译了《Linux命令详解手册》、《编程人生》等图书。网络ID为leal，管理Vim、Android等豆瓣小组，个人站点：http://linxiao.net。

张菁　IT从业者，曾就职于IBM、Rogers。近十几年来所做项目主要是为银行和电信行业开发业务管理软件，用的是IBM小型机。从项目的需求、设计、编码到测试、实施和用户生产机上的实时修正，均有深入涉足。此书乃翻译的处女作品。

吴咏炜　自小喜欢编程，浸淫其中二十余载，仍乐在其中。近年来喜温史事，知古而鉴今，亦人生一大快事。《编程大师访谈录》以程序员为题，又含多位大师之史料，故图灵一邀则应从。虽书老而珍视之，于工作之余参与译介。本人现今主要从事软件架构工作。英文个人网站可参见http://wyw.dcweb.cn。可在Google的各项服务（如Mail、Plus和PicasaWeb）中用wuyongwei找到我。

图书在版编目（CIP）数据

编程大师访谈录 / (美) 拉默斯 (Lammers,S.) 著 ;
李琳骁, 张菁, 吴咏炜译. -- 北京 : 人民邮电出版社,
2012.1
书名原文: Programmers At Work
ISBN 978-7-115-26431-2

Ⅰ. ①编… Ⅱ. ①拉… ②李… ③张… ④吴… Ⅲ.
①程序设计－工程技术人员－访问记－世界 Ⅳ.
①K816.16

中国版本图书馆CIP数据核字(2011)第191353号

内 容 提 要

本书是对19位计算机行业先驱的采访实录，采访对象包括查尔斯·西蒙尼、比尔·盖茨、安迪·赫兹菲尔德、雷·奥奇、杰夫·拉斯金等。访谈涉及他们软件创造过程的灵感、技术、编程习惯、动机、反思，以及对未来软件的畅想等。问答中集结了这些计算机先驱的精辟言论，处处闪烁着智慧的火花。

本书适合IT从业人员阅读。

编程大师访谈录

◆ 著　[美] Susan Lammers
译　李琳骁　张　菁　吴咏炜
责任编辑　傅志红
执行编辑　李　瑛

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：700×1000　1/16
印张：23.25
字数：369千字　2012年1月第1版
印数：1－6 000册　2012年1月北京第1次印刷

著作权合同登记号　图字：01-2010-2359号
ISBN 978-7-115- 26431-2

定价：59.00元
读者服务热线：(010)51095186转604　印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

本书赞誉

“个人计算机软件是如何制作的？为什么这么做？做这种工作的人是不是很像歌曲和故事里那些卡通人物式的呆子？对于想知道这些的人而言，这是一本见解深刻的好书……你可以从中更多地了解这个充满冒险想法和智力挑战的领域，了解程序员们的献身热情和不拘常规的做法。”

——Jim Seymour，联合专栏作家

“Susan Lammers为我们成功呈现了一本独具风格的图书，它是大胆而有价值的‘创意世界丛书’中的一本：对19位明星程序员进行和善而有深意的采访，让受访者自己畅谈……强烈推荐这本书，无论是对计算机迷，还是对厌恶计算机的人，这都是一本极有趣的读物。”

——《计算机周刊》

“读每一篇访谈就像是打开百事食品的‘好家伙’包装盒，其中的深刻洞见解除你一切疑惑，让人惊喜连连。”

——Steven Levy，《全球评论》记者

“Susan Lammers让她的采访对象以自己喜欢的方式表达出谈话的内容，他们的个性和喜好仿佛从每页纸上流淌出来……对于初出茅庐的程序员，本书是一本有关工作习惯的宝贵指南……对我们其他人而言，透过本书，我们有幸认识了计算机界面背后的那些名人。”

——*Macworld*

“这些访谈实用而富启迪作用……可读性非常强！”

——《旧金山观察家报》

“对所有有志于设计、创造最棒计算机程序的人，本书都是必读的！”

——《芝加哥论坛报》

“绝对迷人的访谈实录……任何正式涉足编程的人都能发现这些洞见的迷人之处。”

——计算机图书书评

“程序员们喜欢这本书是显而易见的，但本书的部分章节对那些喜欢音乐、视觉艺术、写作和金融的人来说也是相当有吸引力的。”

——*Pace*杂志

译 者 序

从事嵌入式Linux内核/驱动开发，业余以技术翻译为乐，时而客串编辑，好为爱书挑错，以求完美，却也常因“小”失大，不得读书要旨。

我与《编程大师访谈录》英文原版同属80后，与书中访谈对象却至少差了一代。这次翻译也是自己与这些编程大师“艰难”对话、重温那段历史的过程，所幸有互联网相助，困难重重但也趣味盎然。初审译稿，通读全书，总体来说访谈对象以商用软件尤以个人电脑软件的编程先锋为主，其他如UNIX领域的鲜有着墨。不过上世纪七八十年代正值个人电脑风起云涌之际，个人电脑显然也是与普通大众距离最近的，全书有此偏重，也在情理之中。如欲了解其他领域的编程大师，推荐《编程人生》（*Coders At Work*）一书。



此番参与《编程大师访谈录》，历时一年半，拖延多次，感谢朱巍、李瑛、傅志红等诸位编辑的耐心和把关。与咏炜兄互审译稿，受教良多，又得张伸兄（@loveisbug）指正错漏十数，感激不尽。期间往来邮件数百，字斟句酌，谈笑甚欢，实乃一大幸事。

衷心感谢开心外公外婆对我们的照顾，两鬓泛白，仍为我们而分离两地；感谢女儿开心，和爸爸一起读书游戏成长；感谢开心妈妈的支持和容忍；感谢父母的养育之恩。谢谢你们！

李琳骁

我编程也有十几二十年了，所以去年回国时好友小米觉得俺实在是太闲了，又能与此书产生共鸣，于是就拼命鼓励俺参与此书的翻译。可俺在外漂泊了十余年，现如今中文是提笔忘字，英文看得懂和能翻得明白那可是天差地别呀，从没接触过翻译的俺真是愁啊。

不过当读此书看到这些编程大侠们在做项目时废寝忘食、不眠不休、挠墙捶地画圈圈时，俺平衡了，原来那一代天才大侠们也曾经历过俺编程时所经历的一切，而他们的那些经验之谈也让我不禁会心一笑。做项目很苦，编程很累，养家糊口很重要，但要想编出好程序，真的要有兴趣才行。这些大

师是真的痴迷于此，他们中有些人如老乔一样已然离去，但却留给我们一个从不曾想像到的信息新时代，深深感恩并祝他们在天上过得好！

这十几年来我一直在IBM的小型机上开发商用管理软件，主要用于银行和电信业，其中有不少是和钱有关的记账软件，哎，谈钱真的是伤感情啊！现在大家都在用QQ、Facebook、微博时，我的不少用户还在用Green Screen，而且他们只喜欢用Green Screen，为什么呀？答案是："简约！"就像此书中很多大师所倡导的那样。

在整个翻译过程中，深深感谢我的好友米全喜逐字逐句地校对和审稿。小米，没有你的帮助，参与及完成这翻译工作对我来说会是：Mission Impossible！感谢图灵各位编辑的包容和耐心，也谢谢家人对我无条件的支持和帮助！

张菁



参加《编程大师访谈录》的翻译工作，也算是一个意外吧。不过，当朱巍编辑跟我联系时，我很快就对本书内容发生了浓厚的兴趣。毕竟，近些年来，计算机之外我看得最多的就是史书。而这本1989年出版的书在今天来看，就是一本活生生的历史。我读中学时还用过dBASE，在二十多年后的今天，亲身翻译其作者的故事，别有一番滋味在心头。

当然，这本书不完全是历史和"故"事。里面对时代的洞见，产品开发的过程和方式，放到今天仍然是有意义的。历史不能重复，大师对未来的看法可能也常常犯错，但毕竟后辈是可以从先人的经历中学习的。我想这也是这本书最大的意义。

平时看别人的翻译，总觉得这也不行、那也不是。轮到自己，才知道其中艰辛。幸好有琳骁兄弟和李瑛编辑的把关，才不至于犯下低级错误、误人子弟，成为被自己嘲笑的对象。在此表示衷心的感谢。不是大师，干活确实更需要通力合作才行。

最后，还要感谢老婆大人允许我"不务正业"；请昆昆和宁宁原谅爸爸陪你们玩得不够多。没有你们，纯粹的程序人生还是太无聊了:-)。

吴咏炜

本书翻译分工如下。李琳骁：第1～3、8、11、14、15、16章、词汇表、附录；张菁：第4～6、9、10、13、17章；吴咏炜：第7、12、18、19章。编程大师近况"续写传奇人生"部分由图灵公司编辑整理。

中文版序

要描述技术领域革命性的巨变，20世纪可谓是美国人的世纪。20世纪的最后25年，也是本书最早出版的年代，我们看到了一场惊人的革命。个人电脑（PC）把大型机的威力和连通能力送到了每一个人的手中。新型的软件——第一个电子表格软件、第一套PC和Mac操作系统、第一个字处理程序——推动着这场革命。创作这些软件程序的许多人都出现在本书中，他们进一步创立了大型的技术公司。由PC掀起的这场革命，如今继续在更小却更全能的移动电话和平板电脑上上演，这些设备如今遍布全球各个角落。

那么，是谁在驱动本世纪的这场技术革命呢？谈到对未来技术发展的影响，21世纪也可以叫做中国人的世纪。如今，在技术方面取得巨大进展和革新的聪明的软件工程师，不仅来自美国，而且同样可能来自中国。本书汇编了一系列经典的访谈笔录，其中采访了软件创新产业里的一代天骄，几十年来照亮了全世界程序员们前行的道路。通过这次新推出的中文版，我们希望将创新、灵感和智慧的种子，播撒到今日中国众多朝气蓬勃的编程学子心中。



为什么说这些旧日的访谈到了今天还意义重大呢？我对PC革命先驱的访谈，就是要把你直接带入这些杰出的年轻程序员富有想象力和创造力的大脑中，看看他们是如何思考并迸发出新思想，然后怎样逐步完善，再投入到艰苦卓绝的编程工作中，最终得出大众需要使用的工具。多年以来创新过程的根本并没有什么变化，因此，这些访谈内容在今天看来依然很有价值，发人深省。这些访谈还揭示了伟大的人物是怎么开始创新并改变世界的，通常从一无所有开始，仅凭一支铅笔、一个记事本以及坚定的信念，就此开创出丰功伟业。以我们今日的学识，去看看上世纪80年代的访谈内容，读一下每个程序员的代码和注解，读者定能够充分了解这些PC革命的领袖们如何以星星之火激起燎原之势，在软件业树立起自己的游戏规则。今日的软件行业里，也不断重复着往日的故事。

这些访谈展现了一些业界巨擘的远见卓识、创业激情和编程实践，有比

尔·盖茨、迈克尔·霍利、岩谷徹、加隆·兰尼尔、雷·奥兹，以及其他许多仍然在积极影响软件行业发展的人。虽然说现在看这本书，它只是反映了这些程序员曾经经历过的一个历史时期（甚至有些程序、想法和关注的问题如今已经不再需要考虑），但是访谈内容的精髓依然能够激起全世界追求创新、追求卓越的程序员们的共鸣。我现在还会收到一些读者的来信，倾诉他们怎样在字里行间看到了一脉相承的编程精神、企业创新精神，那些对于经典编程方法的生动探讨，至今仍然深刻地影响着他们的生活。

谷歌研究中心主任彼得·诺维格最近提到这本书时说：

"我不得不说，这实在是一本好书，一本在计算机领域独一无二的好书，这样的书越多越好。有关数学家的小传很常见，有关计算机科学家的传记也不少，但描写编程大师的书却寥寥无几。你如果想了解一名程序员的工作，最好的方法就是去读读他们写的程序。如果没有机会读（或者读过以后），就来读读本书这样的访谈录吧……成为一名程序大师可能要花上十年时间，但细细阅读本书，会帮助你更快地追近自己的目标，至少会让你对身边的程序员们有一个更清晰的了解。"



我希望本书在中国的出版，能够深入下一代有远大志向的程序员的心扉，推进他们的思想和技术生涯向前迈进，为我们的世界带来更积极的改变。也许将来有那么一天，我有机会和来自中国的新一代程序员坐在一起，进行一段精彩的对话，聊聊他们是怎么创造性地工作的，于是我们可以再出些书，记录下各个年代伟大程序员的聪明才智。

感谢图灵公司这些卓越的人士，是他们发现了本书对于读者的价值，并着手让译本得以面世。他们为本书的殷勤付出与不懈努力让我深感荣幸且大为折服。我期待着*Programmers At Work*在中国的印行，内心激动不已。

Susan Lammers

前　　言

对当代最有声望的程序员进行系列访谈的这个想法，是由微软出版社的出版人Min S. Yee提出的。Yee熟悉软件创作过程中的艰难与喜悦，自己又写过几本书，所以当他在微软（主要是一个软件公司，不过也是一个出版社）工作时能注意到作家和程序员工作的相似性，也就不足为奇了。他与微软程序设计人员之间的交谈揭示出了编程的艺术、手艺和科学之间大量细微的差异。Yee发现书店中有无数有关“如何”编程的书，但是以个性化、深入的方式展示软件设计人员的经验、方法和哲理的内容却非常匮乏。所以，微软出版社决定揭示软件背后的思想和个性。而我们认为最好的办法就是让这些人在出版的访谈录中讲述自己的故事。



这些访谈不是为了询问程序员有关项目的秘密，也不是要收集他们对软件业日常进展情况的意见。我们的目的是讨论那些在这个激动人心又快速发展的行业中常常被忽视的、不受时间影响的事情。我们想要揭示开发Macintosh那样的操作系统、Lotus 1-2-3那样的应用程序、《吃豆人》（*Pac Man*）那样的计算机游戏的神秘之旅。我们问了这样的问题：那些想法是从哪里来的？把想法转化为现实有多难？开发大型程序是什么感觉？这是一门艺术还是一门科学？是手艺还是技能？同样的事情还能再做一次吗？

作为采访者，我的目标不是支配、欺骗或操控。我没有想着要教别人怎样说话。我的目标是尽可能不被人注意，让程序员自由地反思、反省，然后用言语把他自己的编程方法表达出来。

在访谈过程中，我尽可能问所有程序员相同的问题，这样以后可以对这些访谈进行研读和比较。我们觉得开放式的常规问题有助于突出编程方法间的相似与差异，让每个程序员的个性和特殊兴趣都显现出来。实际情况也是这样的。有些人，如加里·基尔代尔和巴特勒·兰普森，给出了关于编程理论和实践方法的真知灼见；另一些人，如丹·布兰克林和鲍勃·卡尔，把主要精力放在了对特定程序开发过程的探讨上；还有一些人，如鲍勃·弗兰克

斯顿和加隆·兰尼尔，对软件和微机的未来进行了思考。这些发人深省、内容各异的讨论开始勾勒出当今各种各样、不落窠臼的程序员的群像。

有些时候，我会和程序员们见两三次面，谈上几个小时——沉思、谈论、探索。但是，访谈结束并不意味着整个过程也结束了。我们从磁带上将访谈内容记录下来，编辑、精简，然后返给程序员们，这样他们可以读一读自己说过的话。他们可以对访谈内容再加工，确保能确切表达出自己的意思。

此外，我们还请每位程序员都提供一份与工作相关的样本，可以是一段代码、一个程序、一些程序设计的草稿或是信手涂鸦，这样读者可以看到程序员在纸上写下想法时的风格。我们收到了各种各样的材料，有些具有很高的历史价值。比如，我们从丹·布兰克林那里收到了VisiCalc早期设计的草图。安迪·赫兹菲尔德送来的样本也是别具一格：一个完整的程序，有30多页，是Macintosh计算机上叫做IconBounce的程序。所有这些都深刻揭示了这些人的想法和工作。



世界上的优秀程序员为数众多，他们不会都出现在本书中。所以，这是对当代杰出程序员进行访谈的系列图书的第一本。

在这个不断变化的行业中，每一天都有新的突破，会引入一个创新的软件，会成立一家新公司。行业中，明星起起落落只是一夜之间的事情。要找出所谓世界上最优秀的20个左右的程序员，那是在抓瞎。所以我们选取了具有特定专长和阅历的一群人。访谈对象主要是微机程序员，不过有些程序员在小型机和大型机上也有丰富的经验。

“程序员”一词有不同的含义，所以本书的书名是有点问题的。在软件行业，“程序员”一般用来描述编写和开发能在计算机上工作的软件的人。随着软件行业越来越尖端，软件越来越复杂，软件设计人员和程序人员之间正出现越来越多的差别。针对本书，程序员一词指软件开发人员或软件设计人员，常常但并非总是涉及实际代码的编写。书中的有些人，像巴特勒·兰普森、岩谷徹和杰夫·拉斯金，承认自己并没有参与到程序源代码的编写工作中，他们认为自己并不是程序员，而是软件设计人员。他们可能构思了程序的总体想法、开发了算法、编写了规格说明书、设计了功能，但也许并没有一行一行地输入那些构成程序的代码。把这些各不相同、多才多艺的人归到某一类总是很困难的。

本书努力从大量优秀程序员中筛选出一些人，他们中有些人的照片曾出

现在杂志封面上，而有些人则鲜为人知。他们代表了不同年龄段、经历各不相同的人。有的年纪大些，现在正是40多岁，是他们最早发起了微机革命；有的年轻，他们充满活力、不守旧，正准备推动新计算机革命超越了已往的成就。这里有上班族，像C. 韦恩·莱特莱夫和查尔斯·西蒙尼；也有坚定的独立工作者，如乔纳森·萨奇和彼得·罗伊森；还有热情的企业家，如雷·奥奇、加里·基尔代尔和比尔·盖茨。我们发现有些人能够带给我们许多启发，有些人能让我们幡然猛醒；有些人取得了巨大成功，有些人却不太成功。但是毫无疑问，书中访谈的程序员都是杰出的，我们体会了他们对编程这个创意过程的深刻见解，看到了计算机行业中各种各样的人和经历。

虽然访谈的目的是介绍活跃在业内的程序员，但是本书也成为了一部软件行业的外传，这是由一些主要参与者讲述的。本书的顺序大致反映出这个行业的历史，虽然很多人现在已经转向新的产品，拥有了新的专长。

本书的第一位被访者是查尔斯·西蒙尼。他于20世纪60年代在匈牙利开始计算机研究，那时他用的是一台苏联制造的Ural II计算机。接着受采访的是巴特勒·兰普森，他是查尔斯·西蒙尼在加州大学伯克利分校的教授，也是西蒙尼后来在施乐公司帕洛阿尔托研究中心（下文简称施乐PARC）的工作伙伴。兰普森参与了Alto个人计算机的开发工作，他参与的其他很多基础研究工作都直接推动了微机革命。约翰·沃诺克，另外一位在施乐PARC工作过的研究员，也是PostScript的开发人员，是从犹他大学来到西海岸的。在犹他大学的时候，曾参加了在Evans和Sutherland领导下的计算机图形研究最鼎盛时期的工作。



接下来受访的是加里·基尔代尔，他开发了个人计算机上的第一个操作系统，叫做CP/M。因为比尔·盖茨在BASIC方面的工作，我们接下来采访了他，BASIC是使用最广泛的一种计算机语言。约翰·佩奇，PFS软件产品线的设计人员，是最早一批为商业领域中新出现的个人计算机用户定制程序的人。C. 韦恩·莱特莱夫开发了dBASE，这是最先进的数据库程序之一，在微机革命的很早期就树立起了声誉。然后我们转而介绍了东海岸软件行业的情况，我们找到了丹·布兰克林和鲍勃·弗兰克斯顿，他们是个人计算机上一种最早的电子表格程序VisiCalc的创始人。接下来是Lotus 1-2-3的程序员乔纳森·萨奇，然后我们又与雷·奥奇进行了交谈，他在为Lotus开发Symphony之前，先后在Data General（总部设在波士顿的一家小型计算机公司）为乔纳森·萨奇工作、在Software Arts为丹·布兰克林和鲍勃·弗兰克

斯顿工作。接下来我们采访了彼得·罗伊森，他开发了T/Maker，这是另外一个电子表格程序，比VisiCalc晚6个月问世。再接下来是鲍勃·卡尔，他开发了Framework，那是Symphony的竞争产品。然后我们又把注意力放到了Macintosh计算机上，我们对杰夫·拉斯金进行了采访，他是最初负责Macintosh项目的人。还有安迪·赫兹菲尔德，Mac操作系统的开发要归功于他。

本书最后一组程序员是一些更加喜欢革新的、具有艺术气质的程序员。我们在日本东京找到了极为成功的《吃豆人》游戏的设计者岩谷徹，听他谈了他的作品背后的理念。斯科特·金，一位图形设计人员、音乐人，同时也是一名程序员，讨论了第四方软件和全新的用户界面设计理念。加隆·兰尼尔也是一个音乐人，之前开发过游戏，目前正在参与可视化编程的工作，他认为可视化编程将为我们的计算机体验增加一个新的维度，会给编程带来一场革命。最后，我们采访了卢卡斯影业公司的迈克尔·霍利。他24岁，是本书中最年轻的程序员（同时也是音乐人）。他目前参与的工作是为SoundDroid开发软件。SoundDroid是一种新型计算机，将用于编辑和创作电影中的音频部分。



我们希望本书可以为雄心勃勃的年轻程序员，以及那些希望从专家那里了解到软件行业成功秘密的专业人士提供指导。不过本书不仅仅具有指导价值，它还是一本非常好的读物，它将软件行业幕后的一幅幅场景活化在你面前，并且详述了在开发创新软件产品时出现的大量思想、方法和业界的风云人物。

Susan Lammers

目　　录

```
5200        22 7200 4    5240        16 5006 4
   1        00 0001 0       1        22 6570 4
   2        13 5214 0       2        22 5376 0
   3        22 5311 0       3        22 5306 0
   4        22 5171 4       4        02 7542 0
   5        22 5341 0       5        14 5515 0
   6        22 6631 4       6        21 5123 0
   7        11 0014 0       7        02 5121 0

5210        21 5345 4    5250        16 5012 0
   1        25 2302 0       1        22 5373 0
   2        22 5074 0       2        22 5171 4
   3        11 0002 0       3        00 0000 0
   4        21 5355 4       4        11 0016 0
   5        22 7000 4       5        21 5415 4
   6        56 0000 4       6        22 6570 4
   7        02 5002 4       7        22 5036 0
```

```
5220        16 5006 4    5260        16 5005 0
   1        02 0000 4       1        25 2267 0
   2        22 5332 0       2        22 5074 0
   3        11 0001 0       3        22 5015 4
   4        21 5415 4       4        22 5015 4
   5        25 2304 0       5        22 5134 4
   6        22 5074 0       6        02 5011 0
   7        22 7342 4       7        11 0001 0

5230        00 0002 0    5270        22 5157 4
   1        14 5514 0       1        22 5522 0
   2        21 5415 0       2        22 5435 4
   3        25 2300 0       3        22 6512 0
   4        22 5074 0       4        22 5435 4
   5        02 6077 0       5        22 5022 0
   6        16 5012 0       6        22 7342 4
   7        02 5002 4       7        00 0002 0
```

```
Oct  8 11:59 1985  example1 Page 1


        vfli.dypAfter = 0;
        if (vfli.cpMac == caPara.cpLim)
                {
                int dyp = vpapFetch.dyaAfter / dyaPoint;
                vfli.dypLine += dyp;
                vfli.dypBase += dyp;
                vfli.dypAfter = dyp;
                }
/* First, need to scan thru grpchr, till we find a chr whose ich is >= ichMac
(this can happen because we add a chr and then decide to do the line break
before the character indexed by chr.ich) or until we reach &(**vhgrpchr)[vbchrMac]
*/
        for (pchr = pchrBeg = &(**vhgrpchr)[0],
                (char *) pchrMac = (char *) pchr + vbchrMac;
                pchr < pchrMac;
                (char *) pchr += pchr->chrm)
                if (pchr->ich >= vfli.ichMac)
                        break;
/* Now, enter chrmEnd in grpchr. Note: no need to check for sufficient space*/
        vbchrMac = (char *) pchr - (char *) pchrBeg;
        pchr->chrm = chrmEnd;
        pchr->ich = vfli.ichMac;

        Scribble(5,' ');
        CkFli();
        return;
}
```

上图：1965年在俄制Ural II计算机上编写的代码。所有改动都必须利用goto（以“22”开头的指令）打补丁。

下图：取自Microsoft Word的“匈牙利式”代码。例如，变量名vbchrMac表示该变量为：全局（v），偏移量（b），指向chr结构，当前最大值（Mac，current maximum的缩写）。

变量名chr还有更深一层含义：character run，这是Word特有的。

附录中提供了Simonyi“早期匈牙利风格编码”的示例。



1

查尔斯·西蒙尼



1948年9月10日，查尔斯·西蒙尼（Charles Simonyi）出生于匈牙利布达佩斯。上高中时，他开始接触计算机和编程，父亲安排他给一名从事计算机工作的工程师当助手，当时计算机在匈牙利屈指可数。

1966年，查尔斯高中毕业，同时也完成了他的第一个编译器。凭借开发编译器时积累的经验，他在丹麦哥本哈根的A/S Regnecentralen[①]公司谋得了一个职位。1968年，他离开丹麦进入美国加州大学伯克利分校学习，并于1972年获得理学学士学位，1977年获得斯坦福大学博士学位。

西蒙尼曾先后在加州大学伯克利分校计算机中心、伯克利计算机公司、ILLIAC 4项目和施乐PARC工作。自1981年以来，他一直供职于微软公司。在施乐公司，他开发了Alto个人电脑的Bravo和Bravo X程序。在微软，他组建了应用软件小组，并领导开发出Multiplan、Microsoft Word、Microsoft Excel等广受欢迎的应用软件。

在微型计算机世界的几乎各个领域，查尔斯·西蒙尼都打上了他的烙印，要么通过他自己的作品，要么通过影响和他共事的那些人。他谦逊而活泼，

① 丹麦第一家计算机公司，成立于1955年10月12日。（如无特殊说明，本书所有脚注均为译者注。）

脸上常挂着微笑，几乎能够就任何话题发表评论，不论是否与计算机相关。

我们跟查尔斯见过两次面，一次是在午餐时间，一次是在他的办公室，谈话内容无所不及，从Microsoft Excel的特性，到驾驶直升机，乃至现代诗歌的某些话题。他说话时带有很重的匈牙利口音，这已经成了查尔斯讲话和编程的独特标志。他每天几乎都穿同一身行头，褪色的牛仔夹克、衬衫和破旧的牛仔裤，看上去仍是一副20世纪60年代伯克利大学学生的模样，不过他的学识、举止和成就无不显示出他的过人智慧和丰富经验。

* * *

采访者：*你在匈牙利高中毕业之前就写了自己的第一个计算机程序，是吗？*

西蒙尼：是的。上高中时，我写了自己的第一个程序，还有第一个专业程序。我写的第一个程序是填充幻方，让每行、每列[①]的数之和均相等。我编程用的是一台古老的电子管计算机。一整个下午不停地推按钮才把程序输进那台机器。当天晚上，我头痛难耐，带着几大卷打印有80 × 80幻方的纸回到家里。那是1964年。



采访者：*说说你用过的第一台计算机？*

西蒙尼：那是一台俄制计算机，Ural II。它只有4K内存，支持40位浮点和20位操作指令。这台计算机只能用八进制机器码编程（没有汇编器）。我写了几千行八进制机器码。

这台计算机的操作全部通过控制台完成，你需要自己动手，跟它进行一对一的交互。程序员不必站在一旁等待另一位计算机操作员执行一批卡片。从这个角度看，Ural II酷似个人计算机，因为除了机器和你，不用其他人介入。就4K的内存和缓慢的速度而言，它跟1974年推出的Altair非常相似。1964年Ural II带给我的兴奋就和1974年Altair带给比尔·盖茨的兴奋一样。

显然，Ural II在某些方面有别于个人计算机。Ural II体积庞大，要占用一间很大的房间，输入和输出的方法极为原始——主要是通过控制台开关。控制台看起来像一台老式收银机，上面有整整六列开关，右侧有一个输入键。每一列有8个键，编号从0到7。输入数字的方式同操作收银机差不多。因此，

① 严格来说还包括两条对角线。

要输入2275，你需要依次拨动2、2、7、5这几个键。不小心按错的话，只要还没有按下右侧的输入键，都还可以修正。这种操作非常提神，因为它会伴有大量噪音。每次按动开关都会发出响亮的喀哒声，每当清掉按键时——这全靠机械完成——所有按键一下子同时释放，伴着巨大的啷啷声。

采访者：你的第一个专业程序是什么样的？

西蒙尼：我的第一个专业程序是为一种非常简单、类似FORTRAN的高级语言写的编译器。我把它作为一项创新成果卖给了政府部门，并得到一大笔钱，不过我一分也没花，因为不久之后我就离开了匈牙利。

机遇出现在布达佩斯的一次交易会上，我见到几位从事计算机工作的丹麦人。我跟他们接洽，了解到他们新机器相关的大量信息。在随后一次交易会上，我带上自己事先准备好的一个小演示程序，它能准确反馈任意时刻机器正在分析长表达式的哪一部分。我拜托其中一人把这个程序带回丹麦，拿给他们的主管看。他们肯定很喜欢这个程序，因为他们给了我一份工作。我就这样离开了匈牙利。



我在丹麦干了一年半的编程，攒够了钱去加州大学伯克利分校求学。在校期间，我进入伯克利计算机中心当程序员，挣的钱刚好也够付学费的了。

在伯克利上学时，我写了一个很不错的SNOBOL编译器。有个计算机科学教授，叫巴特勒·兰普森，非常喜欢这个编译器，他还让计算机科学专业的学生在课堂上使用它。后来，他跟另外几个教授一起创办了伯克利计算机公司，我便在那家公司谋得一份工作。伯克利计算机公司倒闭后，核心成员都去了施乐PARC。

采访者：你的编程风格主要受谁的影响？

西蒙尼：影响主要来自两方面——一位匈牙利工程师，一台我在丹麦工作时用的计算机。我在匈牙利的导师是一位使用Ural II计算机工作的工程师。我像个狂热的追星族，卑躬屈膝外加免费跑腿，以换取别人容许我待在一个我本不该待的地方。这不是孩子待的地方。它是全匈牙利（也许）仅有的五台计算机中的一台，被看作重要资产。

采访者：你是怎么个卑躬屈膝法？

西蒙尼：我父亲是电子工程学教授，这个工程师是他的学生。我猜是我父亲

托他帮忙让我进去的。我也尽量让自己能派上用场。我先是给他带午饭，后来帮他拿东西递家伙，最后我主动提出帮他们守夜，看管机器。

他们一到晚上就把计算机关掉，到第二天早上再打开。开关真空管时，电热丝加热或冷却很容易损坏。这台机器有2000个真空管，每次打开时都会坏掉一个。因此，他们上班后的第一件事就是先花一个小时找出那个坏掉的真空管。我在那里守夜的话，计算机就可以一直开着，他们也不用浪费那一个小时。于是，在晚上看管机器的时候，我也就可以用这台计算机了。

总之，我和这位工程师成了好朋友。他是个数学天才。我早年学到的许多技巧都是他教的，有的是关于算术思考，有的是关于符号问题。

另外，那台丹麦计算机对我影响也很大。当时，它拥有的也许是世界上最好的Algol编译器，Gier Algol。去丹麦之前，我已经把这个编译器的全部代码清单研究了个遍。它全都是用机器语言写成的，因此我既学了机器语言编程，又学会了从美学层面上思考编译过程。这个编译器的设计者是彼得·诺尔（Peter Naur）[①]。语法等式巴科斯–诺尔范式（BNF）中的字母N就取自他的名字。我对这个程序知根知底，至今仍记忆犹新。



举个例子，我在伯克利上学时写的SNOBOL编译器只是这个程序的变体。我觉得Gier Algol程序现在仍在我脑海中，也影响着我的编程风格。我总是问自己："如果这是Algol编译器的一部分，他们会怎么做呢？"这个程序真是精妙无比。

有一点我印象很深，就是他们倒着扫描源代码文本的做法。在某些情况下，如果你倒着做事情，之前显得很复杂的问题突然之间会变得非常简单。例如，解析前向引用（forward reference）可能很难。要是倒着扫描，它们就变成了后向引用（backward reference），很容易解析。只要从新的角度看待程序，原本可能很难解决的问题也会变得容易解决。这个Algol编译器处处是玄机。

采访者：你是怎么进入微软的？

西蒙尼：决定离开施乐之后，我开始四处打探。我请鲍勃·麦特卡尔夫（Bob Metcalfe）共进午餐。鲍勃是以太网发明人，3Com公司的董事长和创始人，

---

① 出生于1928年10月25日，是丹麦计算机科学先驱，为ALGOL 60编程语言的创建和定义做出了很大贡献，并因此获得2005年图灵奖。他是目前唯一一个获得图灵奖的丹麦人。

早我两年离开施乐。他给了我一张名单，上面列有我应该去找的人。名单上比尔·盖茨排在第一位。谁排在第二我记不清了，因为除了比尔·盖茨我没再找过其他人。

采访者：编程是一种技巧或技能吗？

西蒙尼：什么是编程？人们对此一直各持己见。有人说它是科学，有人说它是艺术，还有人称之为技能或手艺。我认为这三方面兼而有之。我们喜欢说它蕴含大量艺术成分，但是我们都知道它里面更多的是科学。

孩子们在学校里学习数学，高中毕业时，他们会以为数学就是加法和乘法，甚或代数和微积分。其实，算术，即使简单如加法的运算，背后也有令人难以置信的科学理论作支持。

计算机编程背后也有大量科学理论作支持。例如，哥德尔定理的数学证明冗长而复杂，但是如果借用计算机科学的图灵定理，证明起来不费吹灰之力。信息理论和计算机科学其他领域对数学影响巨大，反之亦然。

编程包含有大量科学，同时，它也有点像手艺。实际上，在许多人看来，编程是一项复杂的技能，这跟工具制造很像，需要精雕细琢。我认为，只要将科学、艺术和技能这三者拿捏得恰到好处，你就能取得一些引人瞩目的成绩。

采访者：你觉得编程的哪部分可以视作艺术？是用户界面设计吗？

西蒙尼：在我看来，编程显然有审美的一面，对用户界面而言，不仅设计中存在，甚至连外观也不例外。当你看到那些丑陋的屏幕时，程序员在艺术上的不足便一览无遗。在其他方面，计算机编程也堪称艺术，正如高能物理也可视作艺术一样。

采访者：审美是只关乎用户对程序的感觉，还是它也直接影响到其他程序员分析该程序并探究其编写方式？

西蒙尼：绝对也会影响的，毋庸置疑。我觉得代码清单和计算机自身的美感一直让我陶醉其中。

例如，那台俄制机器看起来像是科幻小说里的计算机，因为机器里的每个触发器（存储1比特信息的开关装置）都有一个小小的、橙色的老式气体放电灯。数以百计的橙色小灯在玻璃门和柜子后面不停闪烁。机器整个生命



的脉动仿佛就在眼前。

那台丹麦计算机是件精美的家具。它的大小与旧式的衣橱相当。计算机正面有3扇柚木门。有一次，我看到有个美国来的主管半信半疑地盯着机器，就因为它是用柚木嵌板的。它甚至还有一个丹麦现代风格的桌台。整台机器散发着迷人的柚木味道。

伯克利计算机个头非常大，大约有6米长，1.8米高，0.6米深。它隐藏在完全漆成黑色的混凝土穹顶里。它放在穹顶里打着聚光灯的样子看上去有点像电影《2001太空漫游》[①]里的黑色独石。

采访者：当你分析某个程序时，你认为什么样的代码清单或算法结构在审美上是优美或悦人的？

西蒙尼：我觉得代码清单带给人的愉快同整洁的家差不多。你一眼就能分辨出家里是杂乱无章（比如垃圾和没洗的碟子到处乱扔）还是整洁如新。这也许意义不大。因为光是房子整洁说明不了什么，它仍可能藏污纳垢！但是第一印象很重要，它至少反映了程序的某些方面。我敢打赌，我在3米开外就能看出程序拙劣与否。我也许没法保证它很不错，但如果从3米外看起来就很糟，我敢保证这程序写得不用心。如果写得不用心，那它在逻辑上也许就不会优美。



不过假定它看上去不错，然后你打算继续深入。理解程序的结构要困难得多。在结构因何优美的问题上也是见仁见智。纯粹主义者认为，只有那些按照极其严格的数学方式来使用某些很简单的构造的结构化编程，才是优美的。就20世纪60年代之前的情况而言，这种反应非常合乎情理，因为当时程序员并不知道结构化的概念。

不过在我看来，即使程序不遵循这些概念，只要它们有其他可取之处，也可以算是优美的。这就像拿现代诗歌和古典诗歌比较。我觉得古典诗歌很棒，你可以欣赏它。但是你不能只欣赏古典诗歌而无视其他。另外，这也并不意味着，只要在纸上胡乱写上一些字，称之为诗歌，就有了美。但是，如

---

① 2001: A Space Odyssey，一部颇具影响力的美国科幻电影，1968年上演，由斯坦利·库布里克导演。故事根据科幻小说家亚瑟·克拉克撰写的多篇短篇小说的部分内容改编而成，包括1950年的短篇《前哨》，并间接引用《童年末日》中克拉克大力提倡的人类优越论为其题材。

果代码有一些可取之处，我不认为非得是数学意义上的结构化才称得上优美。

采访者：别人读几段你的源代码，有没有可能断定“这代码是查尔斯·西蒙尼写的”？

西蒙尼：噢，是的，毫无疑问。是不是我本人写的可能很难分辨，但有一点是确定无疑的：只要看了代码，你就能知道它是不是我的团队写的，或者是不是受我的影响写的。这是因为我从1972年起写的代码都遵循特定的命名规范，许多人称之为“匈牙利命名法”。你一眼就能分辨出哪些代码是受我的影响写出来的，包括Microsoft Word、Multiplan和Bravo，以及其他许多遵循这些规范写成的程序。

采访者：你提到的“匈牙利命名法”是指什么？

西蒙尼：称它为“匈牙利命名法”是个玩笑。你知道，如果有人说“这对我来说就是希腊文”，这表示他们看不懂，因此也可能它就真是用希腊语写的。这里的“匈牙利”是句反话，因为这些命名规范其实是要让代码更易读。这个玩笑说的是程序看起来这么难读，说不定真是用匈牙利语写的。其实这套规范能够很好地控制程序中所有变量的命名。

要是分解一个程序，把它放进磨床，然后对碎片进行分类，你就会发现程序的大部分都是名字。写下“apples + oranges”，你会发现名字“apples”有6个字符，运算符“+”只有1个字符，名字“oranges”有7个字符，一共14个字符，只有一个字符即加号与运算有关。因此，对我来说，要起到作用或有所改进，合乎逻辑的做法就是尽力完善程序的主要部分，也就是名字。“匈牙利命名法”是一种根据变量的属性自动为其创建名字的命名方法。这跟人们把当裁缝（tailor）的叫做泰勒（Taylor）以及把当铁匠（blacksmith）的叫做史密斯（Smith）非常相似。

因此，面对一个具备某些属性的结构，不要随随便便地取个名字，然后让所有人去琢磨名字和属性之间有什么关联，你应该把属性本身用作结构的名字。这种方法有很多优点。首先，造个名字很容易，想到那些属性时，把它们写下来，名字自然就有了。第二，它很容易理解，因为当你读到某个变量时，从名字本身就能了解到与属性有关的大量信息。这些属性会越来越多，因此很难简明地描述它们。为此“匈牙利命名法”引入了一种缩写符号，以



很小的空间就能展现具体属性。当然，这在不知情的人看来完全是一团乱麻，那个玩笑就是这么来的。

有些人认为，如果他们可以读出代码里的每个字，那么程序就是可读的。实际上，这种意义上的可读性并不可取。没有人会拿着代码清单，站到演讲台上大声朗读程序。关键在于理解。只是能阅读单词并发出音来，这毫无用处。当人们看到采用“匈牙利命名法”的代码清单时，他们发现这些词很难念，可能就会认为代码不是可读的。但实际上，由于名字和属性之间存在关联，它更容易理解，也更便于沟通。那些使用匈牙利命名法编程的人，即使在离开我的部门之后，仍会继续使用它。这种命名法已经打入苹果电脑、3Com及其他许多公司。

采访者：*下面说说你创建程序的整个过程。是否存在适用于所有程序的过程？*

西蒙尼：当然。严格来说，对编程而言，我认为我们应该知道自己想要做什么。如果不知道，那么有一个过程确实是解决各种问题的必经之路，那就是要弄清楚：我试图做什么？目标是什么？



打个比方，我想开发一个菜单驱动的文本编辑器，要求响应速度快，并且提供拼写检查器等。在开始真正编程之前，我需要先弄清楚最终产品。有时候，目标的选择取决于我都掌握了哪些技巧。以Bravo为例，这个程序是以算法为导引的。巴特勒·兰普森描述了两个很有意思的算法，于是我们试图围绕这些算法来编写这个编辑器，以充分利用这些算法。此外，J. 斯特罗彻·摩尔（J. Strother Moore），就是Boyer-Moore字符串查找算法的Moore，在文档编辑方面有几个很有意思的算法。于是我们决定：“嘿，这个编辑器要包含摩尔编辑算法、兰普森的屏幕更新算法还有两个缓存。”等到对目标有充分的把握之后，我才会开始真正的编程。我调整姿态，关上房门，并且大声宣布：“现在我要开始编程了。”

采访者：*当你调整好状态真正开始编程时，第一步会做什么？*

西蒙尼：编程的第一步是想象。就是要在脑海中对来龙去脉有极为清晰的把握。在这个初始阶段，我会使用纸和铅笔。我只是信手涂鸦，并不写代码。我也许会画些方框或箭头，但基本上只是涂鸦，因为真正的想法在我脑海里。我喜欢想象那些有待维护的结构，那些结构代表着我想编码的真实世界。

一旦这个结构考虑得相当严谨和明确，我便开始写代码。我会坐到终端前，或者换在以前的话，就会拿张白纸，开始写代码。这相当容易。我只要把头脑中的想法变换成代码写下来，我知道结果应该是什么样的。大部分代码会水到渠成，不过我维护的那些数据结构才是关键。我会先想好数据结构，并在整个编码过程中将它们牢记于心。

采访者：*这是最重要的一步吗？*

西蒙尼：当然，这是最重要的一步：最优算法的知识当属科学，结构的想象则是艺术。这些算法的细节，以及编写高效代码实现这些结构的转换，是编程像手艺活的一面。从技术上讲，这就是所谓维护结构的不变性。编写代码以维护不变性是相对简单的技艺，不过这需要非常用心并辅之以大量训练才能练就。

采访者：*你对编程感到过厌倦吗？*

西蒙尼：是的。

采访者：*编写程序的过程是痛苦的还是快乐的？*

西蒙尼：两者兼而有之。假装每时每刻都很快乐是做作。就像运动员所说的："要是没受伤的话，肯定是你还不够努力。"二十年后，我已经体会不到刚开始编程一两年时的那种新鲜感。当然，有时我仍会有这种感觉，只不过不像以往那样常有，这是没办法的事。

采访者：*你每天都有固定安排吗？你每天都编程吗，或者你会先把问题放一放，然后集中一周时间搞定它？*

西蒙尼：我不是每天都有机会编程。我不用特意把问题放一放，因为总会有人打断我。我一般晚上编程，白天总是被打断。

采访者：*晚上你会到办公室还是在家工作？*

西蒙尼：我就在办公室工作。我住得很近，非常方便。来办公室就像进自己家另一个房间。我不会窝在家里编程的，来办公室也就是两分钟的事儿。

采访者：*你如何管理手下的程序员？你觉得现在自己做管理多过编程吗？*



西蒙尼：我两样都做，目前还是编程多一些。开发Bravo时，对程序员的管理非常非常直接。有一次，我其实写了一份极为详尽的工作指令，也就是所谓的元程序。这差不多就是个程序，只不过是用非常非常高级的语言写的。我们从斯坦福大学找了两个机灵鬼作为“试验对象”。他们写的程序完全符合我的要求，这样我们实现了双赢：首先，对我来说，用这种非常高级的语言工作更容易，本质上是在对这些人进行编程；其次，他们真正弄清楚了这个程序，效果远远好过我直接交给他们写好的代码清单，并叮嘱他们仔细研读这个程序。他们掌握了这个程序，因为这是他们写的。瞧，每个人都可以宣称自己写了这个程序。这个程序是我写的，也是他们写的。真是太棒了！我认为管理的最佳方法是言传身教，经常复审代码。我们一直坚持开展代码复审。

采访者：*让多名程序员开发一个程序，开发速度会更快吗？*



西蒙尼：不一定。编写同一个程序的人员越多，人均产出的实际代码量越少。结果，总的代码产出一开始会更多，之后实际上可能会减少。以两个人为例，也许单位时间只能多写百分之五十的代码。

顺便提一下，代码的效率还会随着开发同一个程序的人员数量的增加而有所降低。最高效的程序往往是一个人写的。唯一的问题是，它可能需要写上一辈子，而这显然是无法接受的。因此你需要找上三五十个，甚或好几百个人开发一个项目。

采访者：*你能预估编写一个程序要用多长时间吗？*

西蒙尼：预估编写程序要花的时间难度很大。之所以难度很大，原因多种多样。这并不意味着我们就不用尽全力预估，因为预估时间可能用处很大，就像天气预报不仅有经济效益还有其他好处一样。

真正的好程序会永远存在，写起来永无止境，至少只要硬件存在，程序就会存在，甚至更长久。当然，只要Alto计算机存在一天，Bravo就不会消失。编写Bravo的是两个暑期实习生。夏天结束时，其中一人走了，另一个人留了下来。第一个发布版本大概用了3个月。在约5年时间里，一共发布了14个版本。

Multiplan也是大致如此。当你想到Multiplan存在于Microsoft Excel中，

就明白Multiplan将不断延续下去。而Macintosh上的Microsoft Excel也不会是这一链条中的最后一个程序。它会在Windows上延续下去。

采访者：开发Bravo时，你是否认为施乐Alto计算机会成为所有人的选择？

西蒙尼：是的，我当时太天真了。不过，Alto后续机器正逐渐成为每个人都能用的机器，由此可见我的看法并没有错。从某种意义上说，Macintosh计算机和Windows程序都是后继者……（说到这里，西蒙尼停下来接电话，草草地说了几句便接着回来和我聊）。来电话的是汤姆·马洛伊（Tom Malloy），我刚才提到的其中一个暑期学生，留下来的就是他。我有一年没跟他聊过了。他后来去了苹果公司，为Lisa电脑开发编辑器程序。

采访者：你为什么会写程序？你把它看作是工作、职业还是挣钱的手段？这是你与生俱来的能力吗？

西蒙尼：应该是兼而有之吧。我年轻时还算有点天分。即使是在我不懂编程的时候，我就知道与编程有很大关系的一些东西。记住那些复杂的事情对我来说轻而易举。随着年龄的增长，这变得越来越困难。想象场景也不再那么清晰。



采访者：为什么想象场景变得不再那么清晰？

西蒙尼：可能只是年纪大了，思维方式也跟着变了。现在，如果试图清晰明确地想象出包含二三十个组件的东西，我就必须全神贯注，甚至还可能会头疼。换作年轻的时候，我可以想象一座有20个房间的城堡，每个房间里有10个不同的物体，这都不在话下。不过现在我做不到了。现在我更多的是依赖早年的经验进行思考。我看到的是一团团未成形的云，不是像明信片画面那样清晰的图景。不过我的程序确实写得更好了。

采访者：成为优秀程序员有什么套路可循吗？

西蒙尼：恐怕没有。

采访者：这种才能是天生的还是来自后天教育？

西蒙尼：挑选合适的人并培养成优秀程序员，有不少套路可循。我们招揽有天分的员工。我不清楚他们何以有如此才华，我也不用去弄清楚。但是他们

确实很有才。在此基础之上，环境可以起到很大作用。

进入公司第一天，程序员就会拿到几本书。其中一本是数学家乔治·波利亚写的《怎样解题》。（西蒙尼边说边从他办公桌旁的书柜里取出那本书，翻到某一页。）这两页很重要。这本书的其余内容就是基于这两页展开的。这就像一张问题求解的检查单。这是起飞前、起飞和着陆检查单。它不会教你如何飞行，但是如果不照做，即使你已经懂得怎么飞行，也有可能会坠机身亡。

求解问题时，我们遵循以下四个步骤：首先理解问题，然后拟定计划，接着执行计划，最后回顾整个过程。这样的书我们大概有四本，我觉得我们能使程序员比刚加入公司时变得更加优秀。

采访者：你怎么看待将来程序员的作用？

西蒙尼：如果你是在问我们会不会像以前的物理学家那样自高自大，天知道！任何一门学科只要硕果连连，从事那门学科的人似乎总会飘飘然："我们早就知道自己真的很聪明。"然后，他们摩拳擦掌，想要解决其他领域的问题。



这让我想到1945年之后的物理学家。他们说："我们全都搞定了！现在让我们四处看看。"看到生物学与控制论，他们认为："这些研究大脑的家伙什么都不懂。他们甚至不知道记忆是怎么储存的。这也不足为奇，谁叫他们都是呆瓜呢。现在，让我们来研究研究。我们会搞定它的。我们将运用海森堡方程、量子力学，或者以前派过用场的随便什么玩意，我们会把它应用到大脑研究中，接下来就等着奇迹出现吧。"

这有时行得通，有时行不通。天知道。也许计算机科学将会帮助破译DNA，而不是只提供工具。破译DNA可能会成为黑客的终极梦想。

采访者：你觉得将来程序的编写方式会有重大改变吗？

西蒙尼：我认为未来的计算机会比今天的更高效，但我不觉得会有什么大的差别。我不知道第6代或32代计算机会不会做到一些完全不同的或是很了不起的事情。我对鼓吹多妙多好的新方法格外警惕。光从我们现有的方法来看，我就能看出不少改进余地。我更信赖现有的方法，不是因为我保守，而是因为我知道自己至少不会失去既有好处。

我总是担心，当这些所谓难以置信的新好处到来时，以往所有的好处也

将不复存在。然后，它会让你陷入两难境地，你必须自行判断境况是否更好。我喜欢有把握的胜利。我敢断定，改善我们现有的东西，保留既有好处并消除弊端，胜过引入新好处新弊端兼而有之的新玩意。不过也许我是错的，到时我会第一个加入到新事物的行列中。我坚信事情会发生剧变，变得更好，不过这需要时间。

采访者：为什么还要等这么长时间？

西蒙尼：因为必须先等大量愚蠢的想法消逝。这就是为什么进步需要时间。首先，新想法必须不断演变；其次，阻碍进步的坏想法必须消亡。历来就是如此。即使是相对论和量子力学，好的想法也必须经历很长时间才能成形。然后，旧物理学的既得利益者会慢慢消失。

采访者：可以举个例子吗？

西蒙尼：要是提到人们普遍厌恶的打孔卡之类的东西，我估计不会有多大效果。因此，我还是得挑些大多数人相信的东西。我认为"简单崇拜"，即以简单本身作为追求目标的观念，值得高度怀疑。多年来，这个观念一直诱使我们关注那些回报最快的问题。但它只是一种手段。我认为，计算机科学，连同其他所有数理科学（数学、物理和现代分子生物学等），都将通过理解非常复杂的现象而不断改进。数学已经发现了一些复杂的基本对象，在这方面也处于领先地位。有一类这样的对象其传统名称是"简单群"，这颇具讽刺意味地反映出"基本"等于"简单"的旧观念。不过，这两个含义也许并不相同。在计算机中，只是喋喋不休地大谈简单性，我们在实际的人工智能、用户界面和语言等领域也许就会毫无进展。

采访者：不编程的时候，你都做些什么？还有其他兴趣爱好吗？

西蒙尼：还有其他不少有意思的事情，我也乐此不疲。我对埃及象形文字略知一二。学习其他语言、旅行和观察世界都是很不错的活动，我不介意做这些事情。我还持有私人旋翼飞机（直升飞机）飞行员执照。

　　我不觉得编程要比其他事情重要得多。但是如果你从做事的角度来看，它就变成另一回事了。实际上，比起单纯的编程，正是这份业务工作让我忙得不可开交。我非常愿意在工作过程中做编程。

采访者：你是否认为自己更愿意把时间用来完成编程这项业务工作？



西蒙尼：不。我只是说我编程是因为它是工作。不只是因为我喜欢编程，而且因为我喜欢这项工作。我不会一写代码就雀跃不已，说："嘿，我又写了一行代码，真是太开心了；写得越多，快乐越多。"绝非如此。这行代码我也许已经写了10遍。只是不断重复键入，以致弄伤手指，可能会非常倒胃口。因此当我编程时，那么做的原因在于它是工作的一部分，这份工作才是我想做的。

抽象的编程和业务的编程之间，主要区别在于后者目的非常明确。否则，它不过是像下棋一样的抽象活动，游戏终了，棋子乱成一堆，游戏就此结束。程序完成时，有人会使用它，我看到他们喜欢它，自己也会从中获得满足。他们中有些人甚至为它付了钱，我能分到其中一部分，可以用这些钱到埃及旅游，或者飞半小时直升飞机。顺便提一下，驾驶飞机跟编程项目非常相似：起飞和降落很刺激，驾驶过程可能会非常累人，而飞机可能会随时解体。



采访者：你怎么看待自己同时代的其他程序员？

西蒙尼：我非常敬重那些影响过我的人，我非常敬重他们。不过我也非常尊重现在跟自己共事的人。

采访者：你跟其他开发过大型程序的程序员有无来往？你会和他们交流想法吗？

西蒙尼：我非常看重竞争。我有幸在两次展会上遇见鲍勃·弗兰克斯顿和丹·布兰克林（这两位是Software Arts公司共同创始人，世界首套电子表格软件VisiCalc的创作者）。我还遇到过乔纳森·萨奇（莲花公司[1]创始人之一）。但遗憾的是我们来往并不多，这些程序员来西雅图的机会也不多。布鲁斯·阿特维克（Bruce Artwick，微软飞行模拟软件Flight Simulator的编写者，被称为模拟飞行之父）有时会来造访，还有苹果公司的人，比如比尔·阿特金森（Bill Atkinson，Lisa电脑程序员之一，后来为苹果Macintosh电脑开发了MacPaint程序）——在我眼里他是最棒的——和比尔·巴奇（Bill Budge），为美国艺电公司开发了《弹珠台制造机》[2]游戏）。这些家伙都很出色。

① Lotus Development Corporation，1995年被IBM收购后更名为Lotus Software。

② Pinball Construction Set'PCS，一种新的游戏类型，用户可以用它制造虚拟的弹珠台街机。

我们没有太多东西可以高谈阔论。我们都很有默契，说上三两句话就能心领神会。我知道，这些家伙只要开口说话，就会知道自己在说什么。因此当他真正开口说话、也确实知道自己在说什么时，就没什么让人意外的了。而且既然我也知道自己在说什么，说的又是同一件事，那又何必说呢，是吧？这就像笑话专场，听众围坐一圈，说笑话的人甚至用不着讲笑话。他们只要报个笑话的编号，大家就笑翻了。

要是能和这些人一起共事，那就太棒了，只可惜我们却是商业上的竞争对手。我觉得我们聚到一起可以成就一番伟业。也许有一天火星人入侵地球，逼不得已，我们必须实施计算机曼哈顿计划[①]。我们全都会被送到新墨西哥州并肩作战。天知道。

续写传奇人生

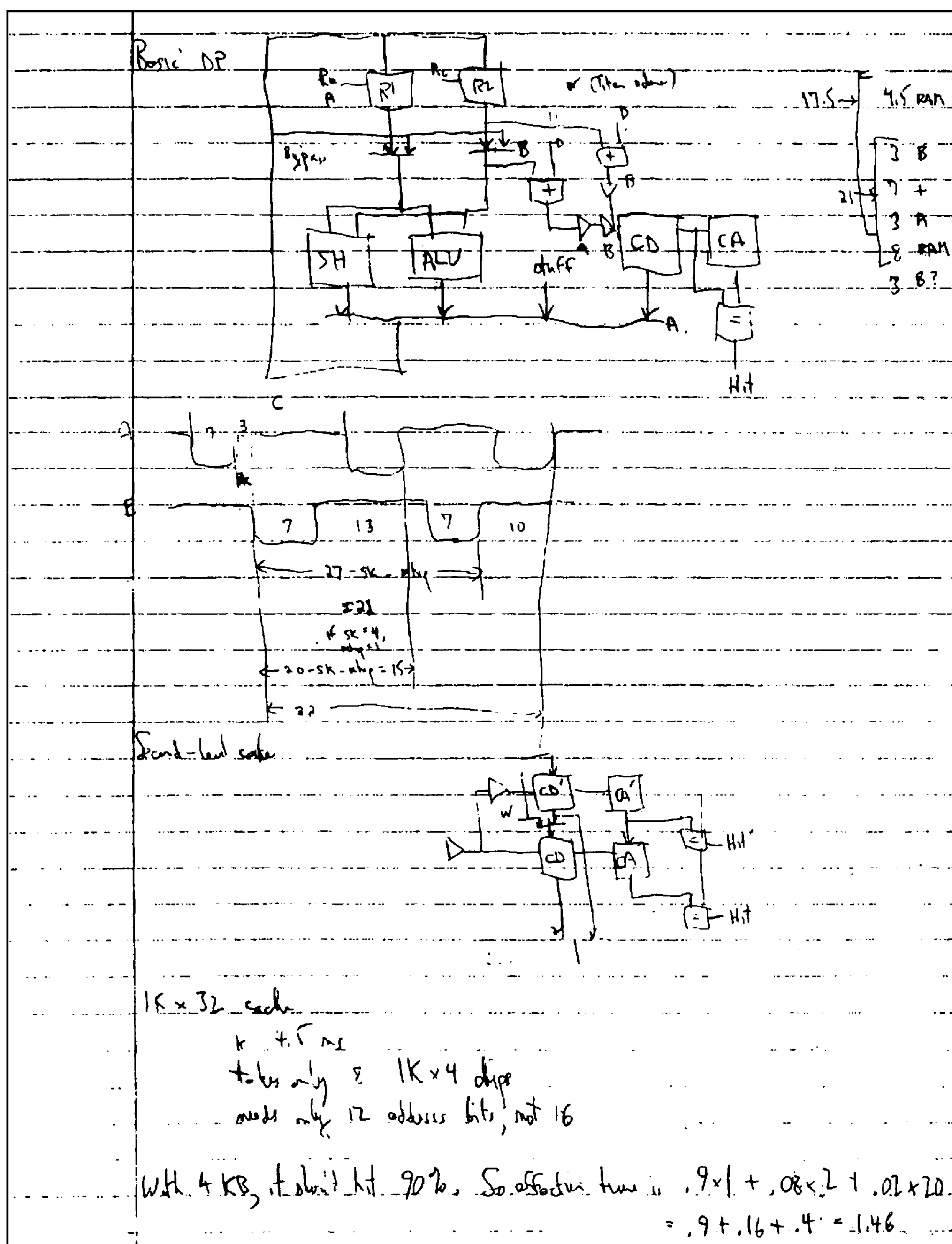


西蒙尼从1981年进入微软，直到2002年离开，当时他在微软公司的头衔是应用开发总监、首席架构师。在微软期间，西蒙尼招聘和管理的开发团队创造了很多最畅销的软件，包括Microsoft Word、Microsoft Excel等。西蒙尼于2002年创办了Intentional Software，目前担任该公司主席和CTO。这家公司的宗旨是创造能加速软件设计的技术，让商务人士即使不熟悉电脑术语，也能清楚地描述需求。

在工作以外，西蒙尼表现出对航天旅行的极大兴趣，并于2006年9月在俄罗斯星城接受训练。（星城是俄罗斯加加林宇航员培训中心的别称。）2007年4月7日，他与两位俄罗斯宇航员一起搭载联盟TMA-10飞船前往国际空间站，并于21日返回地球。4月9日到达国际空间站的时候，西蒙尼说："黑暗天空中的一切都令人惊叹，非常非常激动人心。就像一个巨大的舞台布景，有许多不可思议的歌剧或现代剧的奇妙演出。当我说我彻底折服的时候，就是现在这个样子。"

两年后，即2009年3月，西蒙尼再次进行了太空旅行，重游国际空间站。

① Manhattan Project，美国陆军于1942年6月开始实施的利用核裂变反应来研制原子弹的计划，主要在新墨西哥州沙漠地区的一处绝密研究中心进行。



巴特勒·兰普森绘制的设计草图，一种带二级缓存的快速CPU。该CPU及其他程序的更多规格参见附录。



2

巴特勒·兰普森



巴特勒·兰普森（Butler Lampson），目前在加州帕洛阿尔托数字设备公司（Digital Equipment Corporation，下文简称DEC①公司）系统研究中心担任高级工程师，他曾是加州大学伯克利分校计算机科学副教授、伯克利计算机公司创始人、施乐PARC计算机科学实验室的高级研究员。

兰普森是业界最受敬重的专家之一，在许多计算机设计和研究领域都颇有建树。他开发过硬件系统，如以太网局域网和Alto、Dorado个人电脑；操作系统，如SDS 940和Alto；编程语言，如LISP和Mesa；应用程序，如Bravo编辑器和Star办公系统；还有网络服务器，如Dover打印机和Grapevine邮件系统。

我前往DEC在帕洛阿尔托的办公室跟巴特勒·兰普森见面，他每六周会有一周在那里上班，其余时间则在费城工作。他自称是“电信通勤族”，大部分工作都依赖通信线路完成。

在这个快速发展的行业中，跟其他许多人不同，巴特勒·兰普森对自己创业并未表现出太大兴趣。他的关注点非常单一，就是计算机系统的成功设

① 1957年诞生，1966年8月16日公开招股上市，1998年1月被康柏以96亿美元的价格收购，2001年惠普和康柏宣布合并。——编者注

计，无论是硬件、软件应用、编程语言还是网络。现在兰普森很少写代码，他是系统设计师，利用自身的远见和专长为复杂系统奠定基础。毫无疑问，他是最出色的设计师之一。

*　　　　*　　　　*

采访者：计算机什么地方吸引你？

兰普森：在我看来，计算机是世界上最棒的玩具，你可以利用它把奇思妙想变成现实。

采访者：但在其他领域也能实现这一点……

兰普森：在其他领域实施起来要困难得多。对物理学家来说，只能是大自然给什么就用什么。但是对计算机科学来说，你想要什么只管创造就是。这就像数学，只不过你用计算机创造的是有形的东西。



采访者：你以前研究过数学和物理学？

兰普森：我在哈佛大学学习物理。有个物理学教授想用PDP-1分析火花室[①]相片，临近毕业时，我帮他做了不少编程工作。随后我进入加州大学伯克利分校继续学习物理，当时学校里有个计算机研究项目很有意思，但秘而不宣，不为人知。一次在旧金山参加计算机会议时，我从一个朋友那里了解到这个项目。他问我这个项目进展如何。我回答自己从没听说过，他告诉我，从某一扇没有记号的门进去就能找到它。

采访者：那扇门后面隐藏着什么？

兰普森：那里头正在开发SDS 940，最早的商业分时系统之一。

采访者：你参与这个项目了吗？

兰普森：参与很深。我最后放弃了物理学，因为我发现计算机研究更有意思。选择这行很幸运，因为要是继续攻读物理学的话，我拿到博士学位的时候就会正赶上物理学博士学位大贬值。

采访者：于是你穿过一扇没有记号的门，加入一个保密的计算机项目，从物

① spark chamber，一种利用气体火花放电的粒子探测器，由日本人福井崇时和宫本重德发明。

理学家变成了计算机科学家？除在PDP-1上编程之外，在那之前你还接触过计算机吗？

兰普森：接触过，上高中时，我和朋友在IBM 650上捣鼓过一阵。那台650是最早的商用计算机，快要报废了，因此没有太多租用机时的需求。

采访者：你从事过多个学科的研究。你有没有发现物理学、数学和计算机科学之间的共性？

兰普森：就物理学和数学而言，一如其他正统的学科，要想有所成就，必须能够清晰地思考。这就是计算机行业的许多成功人士都来自这些领域的原因所在。而现在人们通常一直待在计算机系，要有所成就会更加困难，因为这是一门非常浅显的学科，无法驱使你全力发挥出自己的聪明才智。

采访者：你认为这个领域很浅显，是因为它太过原始和年轻吗？

兰普森：这是主要原因。有迹象表明它正逐渐变得不再那么浅显，不过这个过程很缓慢。



采访者：你认为计算机科学与物理学和数学处于同一层面吗？

兰普森：我以前认为计算机科学本科教育很不靠谱，应当予以取缔。最近我意识到这种看法并不恰当。计算机科学本科学位是很不错的专业学位，跟电气工程或商业管理无异。但是我坚持认为，如果你打算在研究生院攻读计算机科学，那么本科学习计算机科学就是犯了大忌。

采访者：为什么？

兰普森：因为从长远来看，你学的内容大部分都没什么价值。你学不到施展聪明才智的新方法，而这些方法对你来说，要比学习如何编写编译器的细枝末节有用得多，尽管后者可能是你本科念计算机科学时的学习内容。我认为，要是所有计算机科学研究生院联合起来，一致决定不接受本科念计算机科学的毕业生，世界会变得更好。学校应当要求这些人补习一年，学习数学或历史等科目，才能继续攻读计算机科学的研究生。不过，我还没看到有学校这么做的。

采访者：是什么样的训练或想法给计算机领域带来最大生产率？

兰普森：通过数学，你学会逻辑推理。你还会学到如何证明，以及怎么处理

抽象要素。通过物理学等实验科学或人文学科，你学会如何应用这些抽象在现实中建立联系。

采访者： 像许多有影响力的程序员那样，70年代初你在施乐PARC智库待过一阵，那里到处都是奇思妙想。对你来说，那段经历是否激励人心？

兰普森： 感觉很棒。我们感觉自己仿佛是在征服世界。

采访者： 当时的同事会影响你的想法吗？

兰普森： 我们全都相互影响。我们经常交换想法，鲍勃·泰勒（Bob Taylor）对我们的影响至关重要。这主要体现在他管理实验室的方式，以及他坚信计算机在某些方面十分重要。

采访者： 施乐公司打造了一个计算机专家的智库，但是许多想法他们都未能实施并推向市场。你有没有因此而失望，你是否认为这个世界还没为这些产品做好准备？



兰普森： 要弄清楚人们有何期待绝非易事。我们意识到外面的世界吗？当然，我们知道它的存在。我们对整个形势了如指掌吗？也许没有。我们对施乐公司卖不动Star工作站感到惊讶吗？不，并不太惊讶。

我对施乐PARC整个事业的看法是，我们不能指望这样的东西永远持续下去。它维持了将近15年，已经非常不错了。

PARC的宗旨在于学习。我们对掏钱让自己学习的公司有所亏欠，所以我们自认为应当在合理范围内竭尽所能，让施乐公司获益。但是，施乐公司培育出那些想法并不是关键。他们的失败并不怎么令人惊讶，因为他们试图进入从未有人涉足过的新业务。公司出问题的原因有很多，部分原因是市场营销出了问题。公司里的技术人员素质很高，但他们一直没找到急需的优秀市场营销人才。

举例来说，鲍勃·斯普劳尔（Bob Sproull）和我花了大量时间设计Interpress项目，那是一种印刷标准。我为此投入了很大精力，原本衷心希望施乐公司接受并促使所有人采纳。相反，他们完全把它搞砸了。结果，参与开发的部分人员离职并创办了Adobe系统公司，开发出一种类似的产品PostScript。很显然，后者成了大家现在都会采用的标准。这种事情很恼人，但是研究实验室的主要产品就是创意。

采访者：现在还有致力于创意的研究实验室吗？

兰普森：我的回答也许有失偏颇。在我看来，最好的研究机构是我目前供职的数字设备公司，另外就是贝尔实验室。

采访者：你觉得研究有实际限制吗？

兰普森：我认为专家系统行不通。70年代初，没人怀疑我们参与的项目不可行。至少，我看不出它们行不通的根本原因。然而现在，我可以看到人工智能系统不可行的很多根本原因。有些人貌似只做了极少量的实验，得出的结论往往含糊不清，而且他们还毫无节制地把这些结论推而广之。

采访者：你认为人们缘何对人工智能的想法如此痴迷？

兰普森：嗯，我不太确定。部分原因是基本的计算机谬论，宣称计算机是无所不能的万能引擎。如果没有明显的证据表明某样东西是不可能的，有些人便推定它必然是可能的。许多人不明白复杂性的后果，认识不到这点，他们很可能会引火烧身。如果他们不愿意听取过来人的忠告，那么唯一的出路就是以身试火，最后灼伤自己。对人工智能感到兴奋的人极少亲身尝试过。

　　我看到过这个问题的极端版本。举个例子，国防高级研究计划署（DARPA）资助了一个项目，据称会使用所有绝妙的专家系统和人工智能技术以及并行计算，用来制造无与伦比的军事设备，比如机器人坦克。他们公布了一项10年计划，其中有一个折页，展示开发时间表和里程碑，标明取得突破的时间点。这纯属无稽之谈，因为没有人知道这些事情该怎么做。有些问题也许会在未来10年内得以解决，但是想要有个时间表！这个念头太疯狂了。世界不会照着你的想法运转。如果你不知道问题的答案，就不可能制定时间表，规定项目什么时候完成。

采访者：你好像非常排斥复杂性。你在设计系统时会力求简单吗？

兰普森：对。一切都应该尽可能简单。但要做到这一点你必须掌握复杂性。

采访者：你在实践中如何做到这一点？

兰普森：控制复杂性有一些基本技巧。从根本上，我会分而治之，把事情分



解开，并准确描述各个部分应该实现什么功能。这会变成接下来如何行事的纲要。如果你还没想清楚怎么写规格，那表明你不明白具体是怎么回事。接着，你有两种选择：要么退回到你真正理解的另外某个问题上，要么更努力地思考。

此外，系统的描述不应该太庞大。也许你必须从多个较小部分的角度来考虑一个大系统。这有点像解数学题：你写的书可以包含许多有用的提示，但不能直接给出算法。

采访者：据悉你涉足过计算机的很多领域。你开发过计算机、操作系统和应用程序。这些都需要不同的训练吗？

兰普森：很显然，想要开发应用程序的话，你需要对用户界面相当敏感，而开发硬件时，这就不见得那么重要了。设计硬件时，你通常更关注自己采用的特定技术所强加的反复无常的限制。没有人知道如何构建真正复杂的硬件系统，因此硬件设计往往更简单。软件则要复杂得多。



采访者：你目前还在编程吗？

兰普森：只是抽象意义上的编程。我没时间再写真正的程序。不过，今年我用半年时间写了一个名字服务器的抽象程序，一共25页。这个抽象程序已被转译成约7千行Modula-2代码，这就是抽象程序和实际程序之间的大致关系。所以我在骗人。我已经有六七年没写过真正的程序了。在那之前，我做过很多编程。

采访者：你以前写的比较重要的程序有哪些？

兰普森：SDS 940操作系统的很大一部分是我写的，另外我还写过两三个用于科学和工程计算的交互式语言的编译器。我写过SNOBOL[①]编译器。此外，彼得·德奇（Peter Deutsch）和我设计了一种编程语言，是C语言的一种前身，并为之编写了编译器。我写过设计自动化程序，70年代初我在施乐公司还写过一个操作系统。

采访者：对你来说，现在开发程序是否比10年或15年前更容易些？

---

① 全称String Oriented Symbolic Language，面向字符串的符号语言，1962和1967年间AT&T贝尔实验室David J. Farber、Ralph E. Griswold和Ivan P. Polonsky开发的计算机编程语言的通称，最终形成SNOBOL4。

兰普森：如今设计程序也许更难了，因为人们的期望值高了很多。但实际的编程则是现在要比以往容易许多。机器的内存容量更大，你不必再挖空心思节省空间。你可以更专注于搞定自己的工作，而不必操心如何最大限度利用有限的资源。这点大有裨益。此外，现在的编程工具也有所改善。不过，由于我不再写代码了，因此对我而言，开发更容易了。

采访者：*你在设计或开发程序时都会历经哪些过程？*

兰普森：大部分情况下，新程序就是现有程序的精炼、扩展、泛化或改良。开发全新程序的机会真是少之又少。通常，我会对现有程序或用户界面中的既有模型做些扩展或改进，然后放进新程序。例如，Bravo来自我的两个想法。一个是关于在计算机模型中如何表示正在编辑的文本，另一个是如何高效地更新屏幕。不过Bravo的基本构思来自一个叫NLS的系统[①]，由SRI[②]的道格拉斯·恩格尔巴特（Doug Englebart）[③]于60年代末实现。NLS支持鼠标操作和全屏幕显示结构化文本。这些想法汇聚到一起便促成了Bravo的开发。

过去，我会在纸上记下笔记，然后开始编程；或者用它吸引其他人，让他们开始编程。现在，我会尽量用相当精确而抽象的语言写出构思的关键部分。通常，这个阶段会有大量迭代。

例如，有一次我跟安德鲁·伯特尔（Andrew Birtell）和迈克·施罗德（Mike Schroeder）一道开发一个名字服务器项目。在施乐，他们已经构建过一个名为Grapevine的系统，这是个分布式名字服务器。Grapevine可以处理几千个名字，当数量超过几千的上限时，系统开始故障连连。做完那个项目后，我们对名字服务器该如何结合在一起有所体会，但是我们要处理数十亿的名字，这又引出了若干重要的设计问题。我确定它的基本元素应该是本地数据库，基本上每个数据库会实现一个相当标准的树形结构名字模式。而所有这些数据库会以松散耦合的方式组合在一起。

这些操作的定义和程序的编写都是概要设计层面的，不太关注细节。我们最终拟定了一份25到30页的材料，结合有程序和规格说明。这份材料被用作详细设计，有个暑期实习生借此写出了七千行代码的原型实现。

① 全称On-line System，在线系统，是革命性的计算机协作系统，由Douglas Engelbart及ARC研究人员共同设计。

② 全称Stanford Research Institute，斯坦福研究所，创建于1946年，原为美国斯坦福大学所属的研究机构，1970年改组为国际咨询研究机构。

③ 人机交互的先驱，以发明计算机鼠标闻名于世。



这就是我遵循的一般过程。需要多长时间取决于问题的难易程度。有时需要历经数年。

采访者：你开发语言也要用这么长时间吗？

兰普森：有时候很快，彼得·德奇和我设计了一种语言，实现第一个基本可用的版本大概用了两个月。但我一直在和罗德·伯斯托尔（Rod Burstall）共同开发一种核心编程语言，一年做上两三次。截至目前，这个项目已经持续了大约5年，我们仍然没有实现。我们的想法总是不断在变。

采访者：你认为要写出优异的程序或系统是否有赖于特定的技术？

兰普森：是的。最重要的目标是尽可能准确地定义系统和外界之间的接口，以及系统自身各主要部件之间的接口。多年来，我的设计风格最大的变化是越来越重视待解决的问题，并寻找准确定义接口的技术。这么做非常值得，不仅能更好地理解程序真正做些什么，还能鉴别程序的关键部分。它还可以帮助人们了解该系统是如何组合在一起的。这才是设计程序最重要的一环。



设计程序和发明算法截然不同。对算法而言，关键是要在头脑中形成整个计划，然后不断重新调整各个部分，最终找到能达成计划的最佳方法。诀窍在于充分明确算法，这样你才能完全掌握这个算法，并了然于胸。

采访者：设计系统或程序时，你依据什么确信它能被实现？

兰普森：一种可能是把设计细化到编程的层面，保证有两点成立。首先，我已经了解编程所用的原语，它们之前业已实现过很多次，因此我对程序可以工作信心十足。其次，我对原语理解得很透彻，足以估算它们会占用多少内存，误差两到三分之一。然后我就可以合理设计自己的程序。这样我就能对实现一个功能信心十足，并粗略估计它的性能。当然，这么做可能会漏掉一些非常重要的地方。这是无法避免的。

另一种可能是非常仔细和正式地写下程序应该具备哪些属性，然后说服自己它的确拥有那些属性。你总是以不正式的方式做这个，但是当你更正式地做这事的时候，就会投入更多精力，遗漏一些要点的几率也会更小。有时候，正式分析会适得其反。但是如果你做得太少，遗漏一些要点的风险就会增加。如果直到程序所有代码都写好了你才发现有所遗漏，那么之前所做的大量工作就白费了。

当然，通常来说你不会扔掉之前的工作，你会试图把它修补好。这么做后果非常糟糕。首先，你并没有解决问题，只是把它转换成另一个问题，而后者不是你真正应该解决的问题。然后你不断修补程序，直到它解决了另一个问题。这种做法很不符合要求。

有时候，我觉得人们想要达成的目标纯属要求过高。程序员往往会无视如下事实：软件系统构建过程中之所以问题频发，就是因为他们尝试要做的东西实在太难了。他们相信计算机是无所不能的万能引擎。这很容易让人误入歧途，以为组建一支1、5、10、50个或1000个程序员的团队就可以让计算机做任何事情。而这显然不对。

采访者：*程序员需要具备什么样的素质才能写出成功的程序？*

兰普森：最重要的素质是能够把问题的解决方案组织成容易操控的结构，其中每个组件都可以用简单的方式说明。有时，成功的程序员可以做到这一点，但他们无法解释自己做了什么，因为他们看不到那个结构。有些人之所以成为出色的程序员，是因为他们可以比多数人处理更多的细节。但是根据这条理由遴选程序员也有不少弊端，它可能导致程序无法由其他人维护。眼下这种现象好像没那么多了，因为现在更流行让一个人或团队设计程序，然后再雇另一拨人编写所有代码。



采访者：*你认为这种做法存在危险吗？*

兰普森：危险在于最终设计人员可能会与现实脱节，从而导致设计无法实现。

采访者：*你有没有设计过无法实现的程序？*

兰普森：没有，我印象中没有。最接近的那次发生在70年代中期的施乐PARC。我和另外几个人设计了一个支持事务处理的文件系统。事务处理就是能以原子操作的方式修改存储数据：要么所有修改全部生效，要么一处修改都不生效。举例来说，从一家银行的账户转账到另一家银行的账户中。你不想让系统处于这样一种状态：钱从一个账户上划走，但没有转到另一个账户。这个系统的早期设计我做了大量工作，系统后来也构建起来了。系统确实能工作，但是大家普遍对它不太满意。

采访者：*你可以描述一下优美的程序吗？*

兰普森：我不知道有无可能回答这个问题。我说不出何为美仑美奂的画作或

者何为美妙的音乐。也许程序描述起来更容易些，因为它包含很多种特性，它既是工件，也是艺术。

优美的程序就像优美的定理，它会优雅地完成工作。它的结构简单明了。人们会说："哦，是的。我知道怎么做了。"

采访者：*如果可以拿编程和另一种艺术比对，比如绘画、写作、作曲或雕塑，你会选哪一种？*

兰普森：建筑也许是更好的选择，它涵盖较多工程上的考虑。能被视为纯艺术的程序少之又少。程序应当实现某种功能，而艺术性只是其中一部分。不过，在我说编程是艺术时，意思跟说数学是艺术差不多——人们通常也不会把数学列为艺术。

采访者：*计算机科学从哪个角度来看算是科学？*



兰普森：最简单的回答是，它是科学，就跟数学是科学一样。计算机程序可以视作数学对象。要理解这种对象，一种方法是对你可以证明的东西做抽象陈述。另一种方法是试错，但效果不是很好。哪怕再不怎么复杂的东西，你也可能尝试很长时间仍找不出对象具备什么属性。

采访者：*你觉得系统的设计和开发方式会发生剧变吗？*

兰普森：怎么说呢，既会又不会。让设计变得更好，关键在于开发更高层次的抽象，然后基于该抽象开展工作。例如，与编写BASIC程序来解决特定类型的问题相比，对VisiCalc电子表格编程会更容易些。

另一方面，我们的期望值不断提高，因此开发更好的抽象并不会使编程任务更容易：这意味着我们可以做更精细的事情。我们可以做更多，因为我们使用的原语更为强大。

采访者：*你是否认为微型计算机和个人电脑只是发展过程中的一个阶段，最终将逐渐演变成其他东西？*

兰普森：当然不是！我给一般听众做过题为"计算机革命尚未到来"的演讲。演讲的基本主题是世界上的重大变革需要很长时间。看看工业革命，从它开始到对人们生活产生重大影响至少用了60年或更长时间。第二次工业革命也不例外。电话和电灯发明于1880年左右，但直到20世纪20年代才得到广泛使用。

我认为计算机革命也不例外。人们倾向于认为我们的前进速度要快得多，因为我们从构思到成品只要6个月。这实在是无稽之谈。计算技术才刚刚开始成为经济的重要组成部分，并开始影响人们的生活。技术发展趋势日新月异，而这些趋势的自然形态便是所谓的个人计算机。

你可以看看基础技术的发展趋势，并观察它的进展。制造计算机要用到硅、旋转磁介质、键盘和显示器。这些组件都在不断演变，你可以在小机箱里实现越来越丰富的功能，而且价钱适中。目前没有任何迹象表明这一趋势会发生改变。我并不是说大型计算机会消亡，而是认为个人计算机的潮流势不可挡。

采访者：你认为未来5到10年这个行业会发展成什么样？

兰普森：至少在未来20年内，计算机行业仍将保持快速增长。目前，还有其他数不清的事情计算机最终会做到，但现在还做不了，只不过是因为它们还不够便宜，速度不够快，或者人们根本还没把问题理解透。

采访者：你觉得计算机科学或计算机行业在我们社会中的地位会改变吗？计算机科学家会像20世纪初的物理学家那样取得重大突破，深刻地改变这个社会吗？

兰普森：不。它不会拥有物理学家在20世纪初那样的地位。大多数人甚至到了1945年才知道物理学家的存在，那一年他们因为原子弹而家喻户晓。我并不认为计算机已经引人注目到足以与之媲美的程度。但我前面说"计算机革命还未开始"的意思是：20年后每个指尖上都将有一台计算机，它们将无处不在，无论其存在方式是否可以预见。这将导致世界运转的方式发生巨变，就像汽车带来的巨大变化一样。但这同样需要很长时间，正如汽车过了很久才改变社会一样。1920年汽车面世时，它的影响还很难估量。有些影响显而易见，但不是全部如此。就这点而言，计算机更是如此，因为它引起的变化将更加深刻。

采访者：你有没有看到20年后每个指尖上都有一台计算机会出现什么问题？

兰普森：我不觉得那有什么问题。显然每只手腕上都会有计算机。计算机的存在是予人帮助，我希望它们会给人们带来正面的影响。

我最近看到一篇很有意思的文章，讲的是加密技术如何让个体更有效地控制个人资料怎么传播。要预测这些事物的实际影响总是异常困难，因为这



不仅涉及技术问题，还牵扯许多政治因素。但这篇文章是个有趣的例证，表明计算机技术可以让个体更好地控制自己的生活。即使在银行等大型机构无孔不入地介入我们生活的情况下，我们仍然可以通过巧妙地使用机器，找到给予个体更多控制力的方法。10年前，没人想象得到计算机会赋予我们个体更多控制力。实际上，当时大家都以为结果会刚好相反。

采访者：*就目前市面上的个人计算机而言，哪些在你看来是问题？*

兰普森：个人计算机还比较蹩脚。我没把这看作是问题。它们是新生事物，人们正在学习如何使用它们，它们也日益变得越来越好。艾伦·凯（Alan Kay）就Mac电脑发表过著名的评论："这是第一台好到值得批评的计算机。"对于正在构建下一代计算机或程序的人们来说，为了做出更好的下一代产品，反思现有计算机或程序存在什么问题显得很有意义。

这也是我为什么会认为计算机普及教育这个概念糟糕透顶的原因。我说的计算机普及教育是指学习使用当下的BASIC和字处理程序。这与现实没什么关系。的确，现在很多工作都要求BASIC编程，但所谓"BASIC是21世纪信息处理社会生存之根本"之说完全是胡说八道。21世纪也许不会再有什么BASIC语言。



采访者：*那么我们应该如何做好准备去迎接未来？*

兰普森：让计算机普及教育见鬼去吧！它实在是荒谬绝伦。学习数学。学会思考。阅读。写作。这些东西会更有持久的价值。学习如何证明定理：过去几个世纪累积下来的大量证据表明，这项技能可以运用到其他许多事情上。只学BASIC编程实在不靠谱。

采访者：*这个行业里的BASIC程序员是不是太多了？*

兰普森：不，我并不认为用BASIC编程有什么特别的坏处。糟糕的是人们杞人忧天，觉得自己的孩子要是不学会BASIC编程就没有前途。他们用不着瞎担心。

采访者：*但是没人能确定哪些技能必不可少。*

兰普森：嗯，有一定道理，但是我们可以窥见计算机科学的演变方向。你可以看看研究实验室中搭建好的系统，从感性上体会这一点。你要是在1975年访问过施乐公司，就能充分体会到1985年高端个人电脑会是什么样子。

采访者：你是否认为终有一天大家都会自己写程序？

兰普森：编程的程度深浅不一。在构建SmallTalk系统时，艾伦·凯说他的梦想之一是孩子们能用它编写有趣的程序。但它实际上并不是这么回事。他说，SmallTalk就像是给孩子们提供许多积木，孩子们可以用积木搭建特定类型的结构，不过很少有孩子能够自己发明拱门。

如果编程只是给电脑下指令，我觉得某种程度上每个人都会。大多数商务人士都会操作电子表格，在某种意义上，那就是编程。我想你会看到更多类似情况。创造性的编程是另一码事。

采访者：作为程序员，你是否觉得自己的工作几乎成了生活的全部？

兰普森：我年轻时经历过那个阶段，但现在不是了。我年纪太大，力不从心了。

续写传奇人生

兰普森从20世纪80年代加入DEC，一直到1995年离开，在DEC公司历任企业顾问工程师、高级企业顾问工程师。1995年，兰普森进入微软，一直工作至今，历任架构师、杰出工程师、科技院士（Technical Fellow），主要致力于反盗版、安全性、容错以及用户界面方面的研究。他是Palladium安全系统的设计者之一，亦曾作为架构师，负责Tablet PC的软件架构。

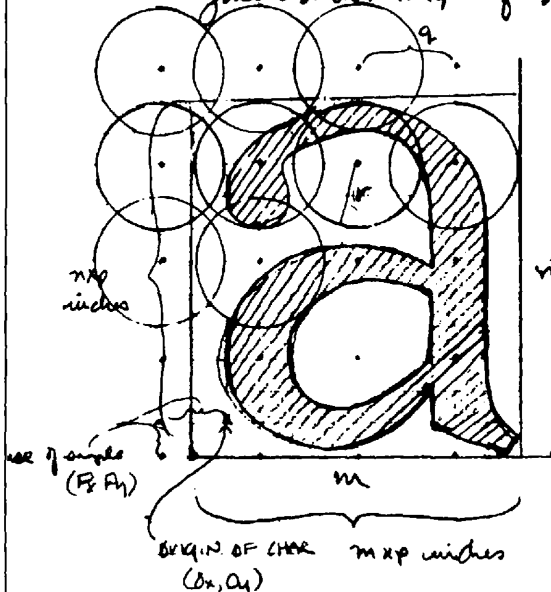
从1987年开始，兰普森一直担任麻省理工学院电气工程与计算机科学系兼职教授。



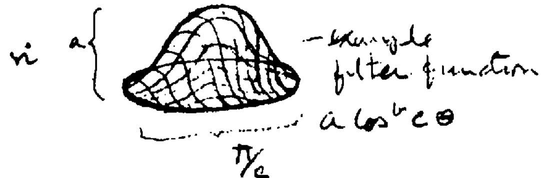
Making Grey scale encoded characters from high resolution bit maps.

The strategy presented here allows one to make grey scale encoded characters from high resolution bit maps.

Given a bit map of m x n bits sampled a p bits/inch



Assume that a sampled image of q inches x at K bits/pixel is desired. Also assume a filter diameter of d (as a percentage of q) is desired



The strategy is to convert all numbers to bit map space so that convolution can be done with indexing.

1.) In bits $q' = q \times p$ represents the spacing between sample points.

2.) $2rq'$ is the array dimension of the filter.

3.) The filter domain (the sample points where the filter has a non-zero intersection with the bit map) is:

$$INT(-rq' - F_x),\ INT(-rq' + F_y)$$

记事本一页，展示如何从高分辨率位图制作灰度字符。



3

约翰·沃诺克



约翰·沃诺克（John Warnock）出生于1940年，在犹他州长大，就读于犹他大学，并获得了数学学士和硕士学位，以及计算机科学博士学位。1968年，戴夫·伊万斯（Dave Evans）[①]在犹他大学组建了著名的研发团队，研究交互式设计和计算机图形学，此时沃诺克刚好转学计算机科学专业。拿到计算机科学博士学位后，沃诺克在加拿大不列颠哥伦比亚省温哥华市创办了一家公司，尝试了一段时间的创业，随后加入加拿大计算机科学公司（Computer Sciences of Canada），在多伦多工作。之后，他前往华盛顿特区，进入戈达德太空飞行中心工作。

1972年，沃诺克搬到加州，跟戴夫·伊万斯和伊万·萨瑟兰（Ivan Sutherland）[②]一起参与Illiac IV巨型计算机、美国宇航局航天飞机太空飞行模拟器和飞机模拟器等方面的工作。1978年，他加入施乐PARC，在计算机科学实验室工作了4年。在PARC工作期间，他致力于提高计算机灰阶成像的排版效果。

---

① 1995年计算机先驱奖获得者，犹他大学计算机科学系创始人。

② 1988年图灵奖获得者，美国科学院和工程院两院院士。他开发的Sketchpad是有史以来第一个交互式绘图程序，改变了人们与计算机的交互方式，奠定了计算机图形学、GUI和CAD的基础。

1982年，约翰·沃诺克博士和查尔斯·戈斯奇克（Charles M. Geshchke）博士共同创建了Adobe系统公司，开发混排文字和图形的软件，其输出与设备无关。他们精诚合作，开发出了Adobe系统公司的第一款产品PostScript。

进入公司办公大楼时，你会一眼看到Adobe公司的标志。这座大楼位于帕洛阿尔托的恩巴克德罗大道（Embarcadero）旁，这条大道两旁散布着不少高科技企业。公司标志非常大，金光闪闪，显示出沃诺克的公司近来获得的巨大成功。跟其他很多人一样，沃诺克带着自己的想法离开了施乐PARC，并在现实世界中将这些想法变成了一门语言（PostScript）和一家公司（Adobe系统公司）。

约翰·沃诺克学者味道十足，他有着满腮的胡子和蓬乱的棕发。他不摆架子，谦逊随和，身着呢绒夹克、开领白衬衫和羊毛长裤。他的办公室让人印象深刻：线条简洁，充满现代气息。看得出来，在我之前，沃诺克已经接待过很多次访问，毕竟他的公司成就非凡。我们寒暄了几句，一起抱怨了一番路途的漫长。目前，他在编程上只做些PostScript的日常维护，不过他依旧是我们这个时代的编程奇才之一。



* * *

采访者：60年代中期，你还在犹他大学时，那里的计算机中心是什么样的？

沃诺克：说来很有意思，因为之前一直在伯克利大学的戴夫·伊万斯从国防部高级研究计划署（ARPA）拿到了一大笔研发资金，每年500万美元，用于研究交互式设计和计算机图形学。他借此吸引到伊万·萨瑟兰、汤姆·斯托克汉姆（Tom Stockham）[①]及其他几位出色的教授，和他一起并肩工作。

原本普普通通的计算机科学系，几乎一夜之间变得光彩夺目。当时我刚好从数学专业转到计算机科学专业。

还在数学系时，我就一直在计算机中心打工，计算机中心主要为大学其他科系服务，比如搭建学生注册系统。有一天，我正在干活，戴夫·伊万斯的一个学生过来问我：“哎，我正在尝试解决这个隐面问题（hidden surface problem），劳烦你指点一下？”我说：“好啊，我觉得这个问题应该这么处理。”我随即开始对它产生浓厚的兴趣，并把自己想到的方法写成代码。我

① 数字音频之父，开发了第一台实用的数字音频录音系统。

的解法较以往必须处理的计算量减少了若干个数量级。出乎意料地，计算机科学系立即拉我入伙，我突然成了他们团队的一员，到全国各地做报告，分享我在隐面问题上所做的工作以及提出的编码方法。

萨瑟兰在1963年就开发了Sketchpad，他的加盟给犹他大学的这个新团体注入了大量活力和创新观点。这个非凡的团体很短时间内就取得了大量成果。埃德·卡特慕尔（Ed Catmull），后来成为卢卡斯影业公司（Lucasfilm Ltd.）的负责人，毕业于犹他；吉姆·布林（Jim Blinn），完成了难以置信的木星飞行器模拟飞行的全部工作，也毕业于犹他；马丁·纽维尔（Martin Newell），后来进入施乐PARC工作，现供职于CAD Link公司，也在犹他待过；还有艾伦·凯及其导师鲍勃·伯顿（Bob Borton）和派屈克·伯德利尔（Patrick Boudelier），他们在帕洛阿尔托研究中心完成了许多研究工作，也都来自犹他；鲍勃·泰勒在犹他待过一段时间，还有亨利·福克斯（Henry Fuchs）。这些人大多都是戴夫·伊万斯的学生。事实上，回顾历史，你会禁不住惊叹，正是犹他这拨人所做的工作，才使计算机图形现在得以应用于卢卡斯影业公司制作的电影中，应用于数字出版物和大量电视广告中。



采访者：戴夫·伊万斯的许多学生后来都成为系统设计师。你认为是什么造就了优秀的系统设计师？

沃诺克：真正的天才设计师少之又少。有些人非常擅长这个或那个方面，但系统设计需要寻求真正的平衡，这儿有一组选项，那儿也有一组选项，需要仔细挑选并精心组合，它们才能够很好地协同工作。许多人会先设计算法，然后围绕该算法设计整个系统。

优秀的系统设计不只是工程活动，更是系统不同组件之间的一组折中和平衡。我认为这种折中和平衡才是系统设计最难的地方。

采访者：PostScript语言最初的构想是什么时候想出来的？

沃诺克：PostScript肇始于伊万斯和萨瑟兰时期，当时我们在给海事学院（Maritime Academy）开发一个海港模拟器。我们必须建立纽约港的数字模型，要涵盖1500幢建筑和油罐区、全部桥梁和浮标等，总之是所有陆上景观。这个模拟器模拟的是从船桥眺望港口的视角。我们需要编写一个庞大的三维数据库，以及大量实时软件，才能使模拟器按期望的方式工作。

我们有一年时间来完成这项艰巨的任务。这是个三维全彩模型。我们确定，最愚不可及的做法就是按照模拟器直接使用它的格式设计这个数据库。

也就是说，让数据库和模拟器绑得太紧密。我们决定创建一个文本文件，然后写个编译器把它编译成模拟器需要的格式（只要我们确定格式是什么样的）。我们那时仍不清楚模拟器最终会是什么样子。

于是我们开始以文本格式搭建这个庞大的数据库。在编写数据库和构建这个大型三维模型的过程中，事情逐渐变得非常明显：与采用基于文本的静态数据结构相比，更合理的做法是使用一种语言。它应该是一种非常简洁、容易解析并且可扩展的语言。这就是PostScript基本思想的由来：为这个三维图形数据库开发一种语言。

采访者：跟伊万斯和萨瑟兰共事之后，你就去了施乐PARC，对吗？你在施乐PARC有什么样的经历？

沃诺克：加入施乐PARC很有意思。我1978年进入施乐，前后待了大概4年。研究中心内部有几个政治派别。一个是以前开发过BCPL的Mesa派，使用传统语言；另一个是人工智能派（AI），使用LISP。



我在泰勒的计算机科学实验室工作。我更喜欢LISP风格的语言，因为它们交互性更强，还有解释器，我喜欢解释环境。但是，身处Mesa派的我使用LISP无异于政治自杀，而且我做的工作也不会被认可。

马丁·纽维尔和我决定用Mesa重新实现E&S设计系统。我们称之为JaM。重新实现这个设计系统大概用了3个月。我们把它改成了一门解释语言，这样就可以拿它当实验台，试验各种新想法。

几年后，当影像科学实验室（Imaging Sciences Laboratory）脱离计算机科学实验室时，JaM已被用作主要开发工具。我们开始改造所有图形程序以驱动显示器和打印机。随后，那套基本语言结构演变成了Interpress，成为施乐公司的打印协议。这一套语言结构又一次也是第三次被实现成PostScript。它们本质上是同一种语言。这门语言实际上已有10到11年的历史，在发展过程中经历了重重考验，许多想法也被多次重新实现。

采访者：是什么促使你重点研究PostScript语言结构的文本打印方面？

沃诺克：嗯，首先，打印需要的语言属性是语法简单直接。PostScript跟FORTH差不多，语法很容易解析。这就意味着，如果我想跟另一个处理器通信，只要通过线路发送串行数据，另一头的处理器就能直接处理这些数据。

语法简单的另一个好处是其他计算机程序生成新程序更加容易。简单直观的语法使得这种语言结构成为打印协议顺理成章的候选者，如果你希望自

己的打印协议基于过程描述，并且是一种编程语言，而非静态数据结构的话。

在施乐PARC，鲍勃·斯普劳尔和威廉·纽曼（William Newman）开发了一种名叫Press Format的格式，由静态数据结构组成。但是他们发现，这并不是应对打印最灵活的方式。要增加一个功能，基本上你必须重建整个系统，纳入更多功能。

我们认为打印协议的可扩展性非常重要。这样一来，如果需要更多功能，你完全可以利用语言自身的机制来构建这些功能，而不是从头开始设计。你无法预知自己将来想要什么样的功能。

因此Interpress不啻为打印协议的上佳选择。有两年时间，查克·戈斯奇克和我一直试图说服施乐妥善处理Interpress，但是很显然，在将它推向消费者的过程中，他们实际上是在摧毁它。他们打算增加若干特性，同时砍掉其他特性，这使得Interpress不仅很难实现，而且难以维护，也不易理解掌握。我们觉得要是换自己做，完全可以做得更简单更合理。

基本上，我们采用了最初简洁的设计，并在多个方面进行扩展，使之成为实用的语言，然后我们便投身印刷业。我们之所以选择印刷，是因为当时所有计算机公司都在试图做打印机产品，并投入大量研发资金，开发激光打印机。但是他们并未取得多大成功，多数产品都不能很好地适应不同情况。



我们觉得把PostScript推销给计算机公司大有机会，因为它跟设备无关。它不依赖特定的机器，因此计算机公司不必更改软件，就可以灵活地采用新技术。我们相信这块市场很大，销售不成问题，事实也的确如此。反之，在屏幕业务上，你要跟其他人的屏幕软件竞争，还要顾及操作系统的方方面面；而打印业务则亮丽、清爽而独立，没有太多羁绊。

采访者：*PostScript语言还在朝什么方向演进吗？*

沃诺克：嗯，对于印刷应用而言，语言已经几乎停滞不前。我认为现在是时候进入屏幕世界，让工作站具备像打印机一样的可编程模型了。为此我们必须对它进行大幅调整，因为屏幕有不同的需求。这些需求包括快速完成特定的垃圾收集操作，而内存管理任务对打印机而言并非必不可少。我们还在写这部分代码，它正在朝这个方向演进。

采访者：*你还在写代码吗？*

沃诺克：哦，当然，差不多每天都写。它在智力上很有挑战性，而且乐趣多多。

采访者：你现在都写什么代码？

沃诺克：大多是PostScript代码，只是让页面比以往更加精致，或者写些简短的辅助函数。我不再插手重要的系统代码。

在设计方面和类似活动上，在整体理念和方向以及事情该怎么做上，我投入了很多精力，但我并不会坐下来，落笔写到纸上。公司里不少人这方面做得比我好很多，在这些方面让他们发挥聪明才智要轻松得多。

采访者：对你来说，现在是不是比10年或20年前更容易写出好代码？

沃诺克：工具更好更强大了。编程环境也更优越。语言更具表现力。现在的计算机性能远比10年或20年前的强大。但是选择更多也就意味着犯更多错误的可能性也越大。

采访者：但是，难道以前积累的所有经验不会使编程容易很多吗？

沃诺克：噢，不是的。其实这是交换——获得这些经验的同时也失掉了年轻时拥有的活力。随着年龄的增长，我也许不会再犯那么多错误，但是我也无法再像以往那样充满活力，精力充沛。



采访者：现在你写代码的方法有无改变？你会采取不同做法吗？你会事先把每件事计划妥当吗？

沃诺克：在动手做任何事情之前，我都会深思熟虑。但一旦开始做事，我就不怕把它扔到一边儿。有一点非常重要，程序员看待一段代码应当像对待一本书的烂章节那样，弃之如敝屣。千万不要过份迷恋一个想法，绝不要固守某样东西以致不能在必要时把它丢掉，这才是程序员应有的态度。

另外，绝不要假设你知道的东西别人不知道。总会有聪明的家伙横空出世，提出更妙的算法，或者想出更简单的方法执行某个任务。这一行的一个诀窍就是尽早认识到这点，迅速采纳并善用之，而不必心存“非我创造不可”的烦恼，非要用自己的方式不可。

采访者：怎样才算是好程序？

沃诺克：对好程序执行性能检查时，它的性能应该呈现扁平状。这表明代码里不存在瓶颈。好程序在于它能保持平衡，而不是靠什么奇技淫巧。

我从1963年就进入这一行，至今已经写了20多年程序。经年累月，你的经验愈加丰富，积累的算法数以百计；你牢记自己学到的技巧、处理过的缺

陷以及误入的死胡同。你记得自己犯过的所有错误、取得的所有成功。可以说，执行特定任务就像是从大杂烩中选定菜目，排出一席美味佳肴。你可以左手一份美味右手一盘佳肴，但搁在一起可能味同狗粮。懂得巧妙搭配食材才称得上好厨师。同样，把程序各个部分妥善组合在一起，才是成就优秀计算机程序员之道。

采访者：你怎么辨别普通程序员和优秀程序员？

沃诺克：很难。面试阶段一般区分不出优秀程序员和差劲的程序员。这就像招揽作家。光从面试来看，你分辨不出某人的写作水平有多好。但是，如果这个人著有多篇获奖小说，你对他的才华就会有更清晰的认识。因此，我们的多数员工都是通过非正式渠道招聘到的。在业内做很长一段时间后，你会认识其中许多人。根据他们在业界的声望，你很容易就能找到自己想找的人。

采访者：有没有什么现成的规则？

沃诺克：我从未找到过什么规则。这个行业里的个性五彩纷呈，各种人才不拘一格。有些人确实擅长某一类编程，而有些人则擅长其他类型。这就像写戏剧和写正剧小说的区别。

采访者：你什么时候开始对计算机和编程产生兴趣的？

沃诺克：我本来打算当一名大学教授，从事数学教学。1963年，研究生快毕业时，我才开始找暑期工作。因为找得太迟，我能找到的唯一一份暑期工作就是到凡士通[1]翻新轮胎。在那里干活又热又脏，而且嘈杂不堪。过了3周，我暗自想："我有数学硕士学位，犯不着干这活儿。"

我到当地的IBM公司求职。我悉数通过他们安排的所有测验，然后他们提供了一份不错的薪酬，超过我的想象。IBM把我送到西雅图和洛杉矶接受系统工程师培训，随后，让我负责盐湖地区两个最大的账户。

做了一段时间，我决定离开IBM，回犹他大学攻读数学博士学位。随后，我结了婚，需要一份像样的工作养活自己，于是进入大学的计算机中心工作。正是这引领我进了计算机科学领域。



① Firestone，凡士通轮胎和橡胶公司，1900年成立的轮胎公司，1988年被日本轮胎生产商普利司通收购。

采访者：*如果重新回到学校，你会专攻数学还是计算机科学？*

沃诺克：哦，我一向喜欢数学，解决问题的过程总是那么带劲。回顾以往，我庆幸自己不是很早就接触计算机。

采访者：*早点学习计算机为什么帮助不大？*

沃诺克：我上了大学，一直念到硕士，因此接受了扎实良好的通识教育。我相信在数学、英语和基础科学方面拥有坚实的基础极为重要。然后，等到读研时，可以尽管放手去学习计算机知识。

如果你真想取得成功，先适应社会其他领域，然后再进入计算机行业，这才是更为可取的成功之道。我喜欢计算机，因为它能让你实现之前只能以数学方式写在纸上的东西。你会得到有形的结果。还有调试程序，真正让程序运转起来，真是令人沉醉其中。最后，你可以给机器设定一组参数，让它提供正确的答案。



这真的很有意思。它给人带来极大的满足感，有点像登山。这就像生活中的许多活动：当你成功地把它做出来，并最终让它工作正常，那种感觉妙不可言，你也会因此而无比快乐。

采访者：*在PARC工作期间，你是否想过自己会拥有现在这样一家公司？*

沃诺克：没有。创业很有意思。第一次打算创业时，我们冥思苦想，想象这个世界可能会用到什么东西，然后得出结论，认为我们应该进入服务业。我们可以打造一款电子打印机，个人电脑拨号连接后就能打印。随后，我们去找Hambrecht & Quist投资银行寻求创业资本，但是他们并不看好服务业务。对于服务业务，风险资本家唯恐避之不及。它需要你精通财务运作，而且赚钱的唯一途径就是特许经营。这块业务难度很大，除非你擅长特许经营，但这并不是我们的强项。

于是我们决定找个更传统的商业计划。我们提出建立一个配有文档编写软件的工作站，连接有激光打印机和排字机，同时销售文档系统。这个商业计划跟ViewTech、Interleaf、XYVision和Texet等几家公司的基本雷同。他们早就有同样的商业计划。

努力了3个月并跟几家大型计算机公司接洽后，我们发现这个商业计划非常愚蠢。我们必须建立市场销售渠道和小型制造厂。基本上我们必须构建整个业务链。显然，我们在这个领域并无专长。

不过，我们在打造计算机公司亟需的专门软件上确有专长。于是，我们调整商业计划，转变成一家软件OEM供应商。这样我们就不必介入制造、营销或者发行环节。同时，我们还可以服务于更广泛的计算机市场。最后结果很不错。

采访者：你觉得将来的计算机会跟现在的大同小异吗，还是会大不相同？

沃诺克：考虑到过去四五年来的变化幅度，我觉得很难做出预测。这个时期技术发展速度惊人。例如，5年前你用的芯片只有64K，而现在我们谈论的都是兆位大小的芯片和CD光盘。我看不到发展速度有放缓的迹象。当下没人能预测下一个创新会是什么。信息时代就是一个巨大的反馈环路，你用的每个新工具都可以帮你构建更强大的工具。因此，要是有什么不同的话，那就是未来可能会有更具爆炸性的增长。

采访者：你认为未来计算机在社会中会扮演什么角色？

沃诺克：计算机普及教育肯定会变得不可或缺。人们每天都会跟计算机打交道，只不过没意识到而已。用户界面也得有相应的改善，这样人们才能更有效地使用机器。上述这些会齐头并进。当今社会完全依赖于技术，将来也是如此，我找不到有什么理由会改变。



采访者：你认为出版业会有什么变化？

沃诺克：出版业采用技术创新的速度远远出乎我的预料。我原以为整个过程需要历经数年时间才能成熟，电子出版技术经过缓慢演变才会投入使用。但它投入使用的速度快得不可思议。

大型出版商迅速采用了这项技术。Knight-Ridder出版集团拥有大量激光打印机和Macintosh计算机，Gannett报业集团已购入许多激光打印机。Hearst新闻集团也买了一些，最近我们听说美联社现在利用通信线路发送PostScript程序。这真是令人欣喜若狂，因为激光打印机进入市场才短短几个月。我原以为市场接受PostScript还需要若干年而不是几个月。

现在，许多企业正在使用Macintosh计算机和LaserWriter打印机制作内部时事通讯，而不是依赖传统排字机。人们看到了新技术的好处。办公市场，即传统的字处理市场，正在快速转向激光打印机，以实现更高品质。

采访者：你认为激光打印机的输出质量要过多久才能跟排字机媲美？

沃诺克：还要过一段时间。碳粉颗粒的某些物理特性以及其他问题使它难以

获得很高的分辨率。排字机属于精密设备，输出字体边缘非常锐利，打印机要达到这种效果异常困难。

静电复印需要解决碳粉颗粒大小相关的基本问题。它处理的是带电微粒，所以这实际上事关带电单元可以小到多小，同时仍能吸住碳粉。现在的打印质量不错，但是如果用放大镜比较排字和静电复印得到的页面，你就会发现静电复印跟前者还是没法比。也许某些热敏技术和喷墨技术会先于静电复印术取得成功。

排字机会更便宜，静电复印的质量也会更好。另外静电复印也会更加便宜。过不了两三年，你就会看到2000美元以内的激光打印机。我认为打字机使用的等宽字体终会被高质量字体取代，至少我希望是这样。

采访者：在这一行要取得成功有什么秘诀？

沃诺克：要想取得成功，你需要找一拨才华横溢的人，而且他们的技能可以很好地融合互补。这就是成功的秘诀。



采访者：你编程有什么诀窍？

沃诺克：我不知道能不能归结到几点上。我前面提到过一些。不要早作绑定，尽可能推迟决定时间。眼界放宽一些，设计要比你自认为需要的程度更加灵活，因为从长远看你最终会需要这样。快速让某样东西工作起来，然后还能弃之不用。

从小的开始实验而不是大的入手学习。不要一头扎进周期长达两年且中间不出什么成果的开发当中。最好每两个月就要出点成果，这样你才能进行评估、重组和重新开始。

程序员经常在一开始时过度定义他们的方法。他们可能会从一个中心构思着手，从第一天就开始编码。然后他们发现自己陷入重围，每件东西都开始膨胀，因为它们依赖于太多其他因素。应当反其道而行之，如果让过程较为宽松，保持一定的自由度，并在最后阶段加快速度，长远来说，你会做出更好的产品。

采访者：你认为自己为什么会从事软件而非硬件开发？

沃诺克：也许是我拥有数学家的特质，我偏爱抽象的东西。我更愿意折腾不同的想法以及想法的组合，而不是实物的组合。我对机械不怎么在行。

采访者：编写软件在哪些方面吸引你？

沃诺克：它无时无刻不充满挑战。我觉得编写软件好比著书立说，就像常规写作一样。你努力把想法和观念融合在一起，力争让其他人觉得那是全新而激动人心的。软件业就像出版业一样。你交换的是想法，这就是你的商品。我喜欢编程是因为自己还比较擅长这个，一路走来，我成功地提出了不少新想法。这其中真是乐趣无穷。

采访者：你会把自己哪些想法归类为软件方面的新想法？

沃诺克：最明显的是我在犹他大学最早做过的线消隐工作。这在当时非常新颖，很多程序都是以此为基础开发的。解决看似很难的问题，并使它看起来容易处理，我真的从中获得了很多乐趣。这是伊万·萨瑟兰的过人之处。他能够剖析任何问题，把它分解开，并使它看起来容易处理。他精于此道。

我也乐于见到人们使用我创造的东西。在施乐期间，看到市场上不断涌现出很棒的产品，而我知道自己原本也能做出来（但却没机会去做），这让我异常沮丧。现在，随手捡起一本杂志，快速翻阅，看到LaserWriter印制出来的图片和广告，真是无比快乐。看到别人用上自己的劳动成果，我深感荣幸。这真是人生一大快事。



采访者：你平时都忙些什么？主要在公司还是在家里办公？

沃诺克：我在家办公，不过也会到办公室，另外还经常出差。我会同客户沟通，在会议上发表演讲。目前我负责一些内部事务和许多外联工作。

采访者：拥有自己的公司对你的生活影响大吗？

沃诺克：是的。在最初的一年半里，我几乎全身心地扑在项目开发上，要让各个部分运转起来。目前这个项目已经进入市场营销、公共关系和客户支持阶段。我的日常工作主要包括与其他人见面，跟企业打交道，找那些公司的副总裁交流，并与他们签署合同。不过这确实很有趣。今年跟去年有点不同，去年跟前年比也略有变化。这就是生活。

采访者：PostScript开发阶段，有多少人参与开发？

沃诺克：PostScript开发大概用了20人年。为支持所有字体，代码量变得非常大。LaserWriter的只读存储器（ROM）有512KB，我们全都用掉了，不容出现丝毫差错。出厂的每台机器都固化了这个程序。我估计这是ROM里装

过的最大单块代码之一。在测试过程中，我们一直提心吊胆。

采访者：PostScript开发过程中面临的最大问题是什么？

沃诺克：诚如多数项目那样，最后的2%要占用50%的时间。把代码完全整合在一起要用一段时间。然后，你还必须确保所有通信协议都能工作，可以收到所有中断，一切正常，它不会崩掉。

采访者：一般说来，你们会偏爱某种语言吗？

沃诺克：我们这里的开发人员经验丰富，他们不会偏爱某一种语言。我们会使用最合适的环境和编译器工具集，这些才是需要考虑的因素。计算机语言不过就是计算机语言，一旦你学会了几种，其他学起来也就不会太难了。

采访者：现在的设计师使用的语言屈指可数，你认为是什么原因？

沃诺克：没有动力去使用各种各样的语言。有些人会采取特定的设计路线，如Mesa就采取了独特的设计路线。不过，他们觉得问题主要在于如何把三四十个程序员召集到一起编写一个大型系统，因此它的设计宗旨是要帮助这一过程。如果你相信好软件是以那种方式写成的，那你可能就要设计一种那种方式的语言。但是，我坚信好软件出自2到4人的小团队，成员之间互动频繁密集。最好的代码出自这类努力和付出。我会尽量使系统用不着有20人参与。



采访者：这是因为有太多不同的想法吗？还是因为完成一个设计的方法太多了？

沃诺克：现在样式和接口太多了。我曾经听人说过，一个人写的所有程序无不反映出此人工作的组织。在PARC，局外人很难相信，计算机科学实验室成员之间都是平级关系，鲍勃·泰勒身兼主管。但是他们得给这个多样化的群体找个共同的工具，因为有近30人在设计这个庞大的编程系统。因此Mesa及其编程环境模仿的是PARC的工作方式。

Adobe公司一开始规模很小，只有6名员工。代码是6个人写的，这从代码结构就能看出来。某部分出自甲，某部分出自乙，各有各的特征和接口。相反，IBM是个庞大的组织，他们的代码错综复杂，从其中自我反馈和不同的策略中能看出该公司各个独立部门的影子。有条相当标准的规则是这么说的，如果你想保持某样东西简单，那么开发它的组织也得简单。

采访者：小公司在逐渐发展成大公司的过程中会碰到什么问题？

沃诺克：诀窍是学习惠普的做法，把公司当作20家不同的小公司。不断拆分，决不让它发展成庞大的组织，直到没有人嫌它大为止。在工作关系上，员工数量尽量少，工作内容尽可能专，并以项目为导向，这样他们才能达到最佳工作状态。这也是我的目标。

采访者：在计算机出现之前，数学家都在做什么？

沃诺克：大量的数学运算！其实，我真不知道数学是不是还跟10年前一样好。世界上数学家的数量可能已大幅缩减。计算机相关的部分早期理论、最好的理论成果似乎都是50年代完成的。最好的计算数学都是在计算机出现之前奠定的。在计算机能做和不能做什么事情上，数学家已经为其设定了基调。我不知道现在计算领域还有没有出现什么新的理论成果。我猜测也许有，不过不会是早期那样的重大突破。

采访者：你认为计算机科学是科学吗？

沃诺克：不，不见得。比起科学，它更是工程学科，一门精彩纷呈、硕果累累的工程学科。在我看来，科学就是提出假说，做各种实验，在现实世界中创建模型。计算机却不是这样，它们是关于现实世界模型的自我实现的预言（self-fulfilling prophecy），是极好的信息工具，是当代社会的重大成果，用来操纵和控制信息。但是，说到计算机科学，我不知道它试图追求什么样的真理。



续写传奇人生

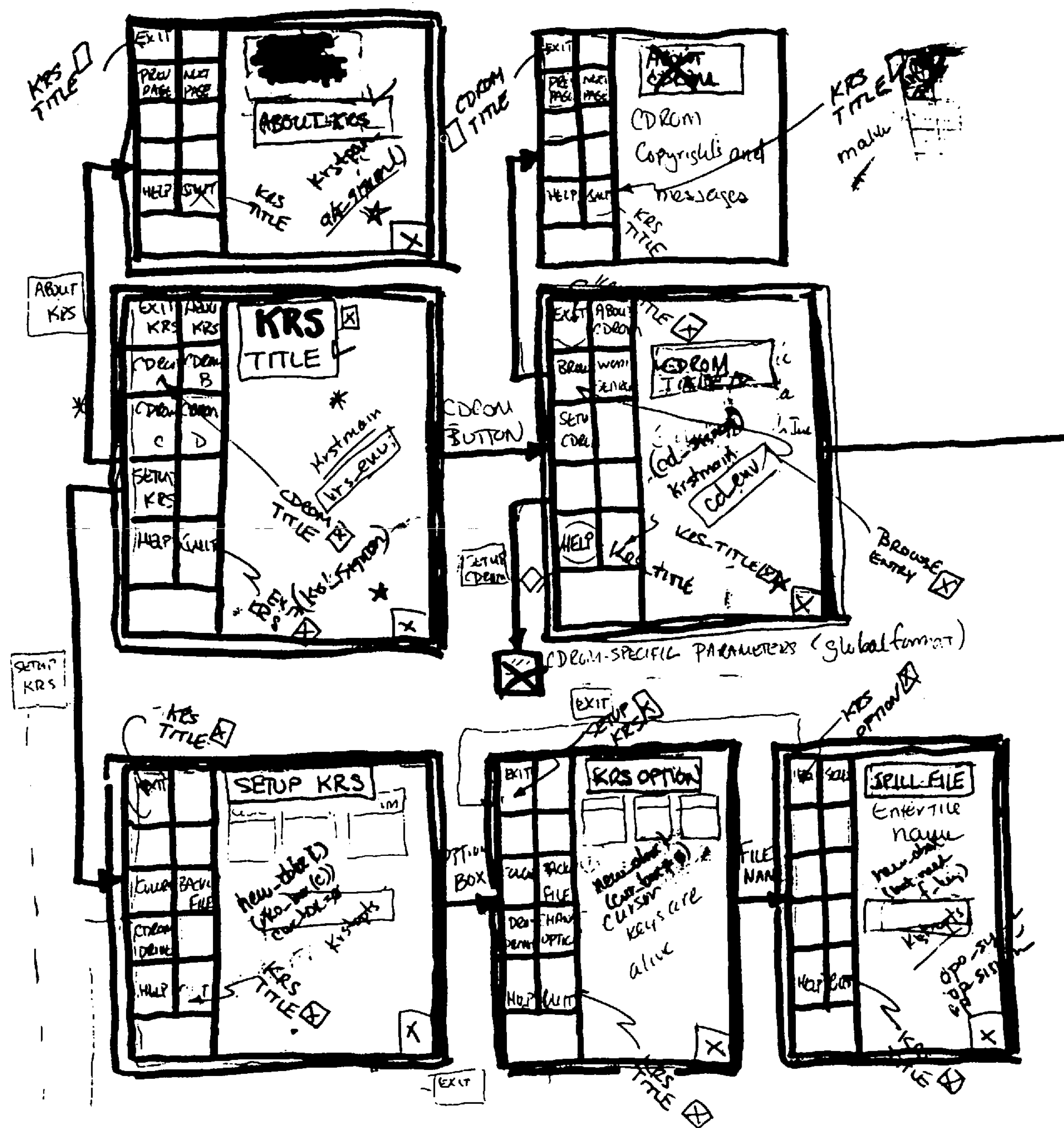
当时Abobe刚刚创建4年，沃诺克担任CEO。1985年，Adobe开发的PostScript与苹果公司面向大众市场的LaserWriter激光打印机结合，组成了世界上第一个桌面出版系统。Adobe的PostScript技术使得从电脑打印文字和图片变得非常简单，从80年代开始，对媒体和出版业产生了革命性的影响。1991年的时候，沃诺克提出了一个名为Camelot的系统，后来发展为大名鼎鼎的可移植文档格式PDF，PDF目前已成为电子文档的标准格式。从PostScript的发明开始，Adobe逐渐变成电子成像和排版领域，以及图形软件产业的领导者，成为全球最大、最著名的软件公司之一。沃诺克担任CEO直到2000年卸任。他在2001年出任CTO，之后则与创业伙伴Geschke一起担任董事会主席，继续为Adobe掌舵。

除了Adobe公司，沃诺克还曾担任Ebrary、MongoNet、Knight-Ridder、Netscape等公司董事，目前仍然是Salon媒体集团董事会主席。他亦曾是圣何塞市创新科技馆的主席，目前还在美国电影协会和圣丹斯协会担任理事。

沃诺克的个人爱好很多，包括摄影、滑雪、网页开发、绘画、徒步旅行、收集罕见的科技书籍和印第安人用具等。沃诺克还和妻子在犹他州的Midway小镇开了一家古典风格的旅馆——蓝野猪（the Blue Boar Inn & Restaurant），这家旅馆坐落在宁静而美丽的希伯山谷边上，距离滑雪胜地Park City、Deer Valley和Sundance都只有20分钟的车程。

2003年沃诺克和妻子一起，向母校犹他大学捐赠了20万股Abobe公司股票（时值570万美元），以建立一幢新的工程馆。该馆目前为犹他大学工程学院主楼。





加里·基尔代尔的这张草图是在开发知识检索系统期间画的，它提供了一个树形菜单设计的平面图像。附件中有关于这个菜单的更多细节。



4

加里·基尔代尔



作为数字化研究所（DRI）的创始人和董事会主席，加里·A. 基尔代尔（Gary A. Kildall）于1972年到1973年间开发了第一个微机操作系统。他把那个系统称为CP/M（控制程序/监控程序）操作系统，后来成为他们公司的第一款产品。此外，他为IBM个人计算机设计了DR Logo编程语言，并开发出微机上最早的高级计算机语言之一，PL/1。

基尔代尔是西雅图人，生于1942年5月19日。他于1972年获得华盛顿大学计算机科学博士学位，随后加入了海军，并且执教于加利福尼亚州蒙特雷的美国海军研究生院，讲授计算机科学。从海军退役后他仍旧在那里继续执教。

基尔代尔于1984年创建了一家名为Activenture的新公司（最近已更名为KnowledgeSet公司），以探索光盘出版业的潜力。Activenture公司于1985年宣布，他们将出版光盘版的《格罗利尔多媒体百科全书》。基尔代尔在担任KnowledgeSet公司总裁的同时，仍旧担任着数字化研究所董事会主席一职。

在一个周一的早晨，我们沿着加州海岸线的1号公路去了数字化研究所。这一路上的有些景色可称得上是全美最壮观的，海岸线旁边星罗棋布的度假村比从事计算机相关业务的公司还要多。数字化研究所和KnowledgeSet公司都是由基尔代尔创建的，它们是该地区中少数几个高科技公司中的两家。我

不禁在想，明知那绝色美景就在门外，又怎能把自己关在那样灰蒙蒙、灯光昏暗的办公室里写源代码呢？但这对专注于工作的基尔代尔来说似乎并没有构成困扰，而在休闲的时候他也懂得如何欣赏周围的美景。他是个工作狂，同时也特别爱玩。他的“玩具”包括一辆新的超级跑车兰博基尼康塔什、一架皮茨特技双翼飞机和一辆劳斯莱斯。

中午时分，加里来到会议室和我们会面。他身材高大，长着棕红色的头发，留着整齐的小胡子，身穿崭新的西部风格牛仔裤、白色牛仔衬衣，脚上套着靴子。刚刚从塔霍湖渡周末回来的加里坦言自己吃得太多了，他要了无糖百事可乐，而我们则要了三明治。接着加里就以其稳重和精于思考的方式，极其认真和热情地与我们讨论起了编程。实际上，为了这次访谈， 我们把他从他最新的项目、光盘版《格罗利尔百科全书》的编码工作中拉了出来。他创建KnowledgeSet公司的原因之一就是为了回归到具体的编程工作中，同时远离他的第一家公司——数字化研究所的管理工作，那个公司已经变得很庞大了。他给我们解释了为了让计算机运转起来而编写代码的整个谨慎而富有创意的过程，在解释的过程中，他常常在白板上勾勒图表或说明要点。



*　　*　　*

采访者：*你曾在海军研究生院教过课。如果再回去教的话，讲课方式会有不同吗？*

基尔代尔：应该不会，因为我现在的编程方式与我当年教课时的编程方式没有任何区别。我最喜欢教的课程是数据结构。它把人带回到了编程的基础，即如何简化问题。编程过程中的部分工作就是解决问题。对于一个复杂问题，不论是设计计算机程序还是建造一座建筑物，解决问题的方式是什么？刚开始你会觉得问题太难解决了，随后你把问题分解成一个一个的小问题。这就是我想教给学生们的。

采访者：*讲授解决问题的原则是不容易的。你是怎么做的呢？*

基尔代尔：在上数据结构课的第一天，我说：“我们来做一个小测验。把你们的书放在地上，拿张纸，写一个程序以符号方式求解微分方程式。给出一个多项式，程序就应当对多项式求微分并产生符号的、而不是数字的结果。”于是学生们就开始写啊、想啊、挠头啊。过了十来分钟，我让大家停下来。

我要求他们思考自己是如何解决问题的。用的是什么工具？已经开始写程序了吗？考虑数学了吗？开始写下一些小例子了吗？整个学季，我们都在探讨解决问题的技术和工具。然后在期末考试的时候，我给他们出了同第一天上课时一样的问题让他们解答。

采访者：学生在上完你的课后，必须学会的最重要的原则是什么？

基尔代尔：我教的两件事情对学生们的学习来说是非常重要的：解决问题以及如何学习。知道如何学习能够让你顺利通过考试并获得其他的校园生存技能。而如果能学会如何解决问题，你这一生过得应该都会不错。

采访者：你自己独特的编程风格是什么？

基尔代尔：我遵循非常明确的、适合自己的流程，虽然这些流程可能并不适合别人。我会先画数据结构，然后花很长时间思考数据结构。在开始编程前，我会一直思考整个程序的流程。

程序就像机械装置一样，一段代码和一段代码之间的工作方式与一个齿轮和一个齿轮之间的啮合方式非常相似。编写程序有点像是建造传动装置。我在几年前写的PL/1语言的编译器就是一个很好的例子。人们都说在微机上编写编译器是不可能的，但经过了几年的运行，它被认为是现有的最优化的编译器之一。

在确定数据结构之后，我就开始写一些小段的代码，并不断地改善和监测。在编码过程中进行测试可以确保所做的修改是局部的，并且如果有什么问题的话，能够马上发现。整个迭代改进的过程是需要速度的，至少对我来说，一个快速的编辑、执行和调试周期是非常重要的。这个方法不能用在大型机或穿孔卡批处理系统上，因为在那上面无法做一些小的修改并查看结果。

采访者：你喜欢在解释器的环境中工作吗？

基尔代尔：不，我不是很喜欢现有的解释器。我希望能有一个像C语言那样的系统语言，能够与现有的编译器相媲美。但它的运行效果怎么样，这还是个问号，因为大多数系统程序都是以性能为导向或是依赖于分时的。如果能有一个非常有效的解释器——就像我在开发PL/M时所用的，或者像现在的C语言，那还值得一用。



采访者：你对编程的兴趣是如何产生的？

基尔代尔：我本来打算当个高中数学教师，并为此在华盛顿大学选修了数学课程。但我的一个朋友给我看了一张FORTRAN语言的指令卡，告诉我那会成为一个非常了不起的东西。我被迷住了，不由自主地产生了兴趣。于是我选修了一门汇编语言程序设计的课程，紧接着又选修了FORTRAN语言的课程，我入迷了。我发现自己喜欢编程的原因就像我喜欢做模型、玩汽车等事情一样。我发现开发程序具有类似的体验。

采访者：你还记得你写的第一个程序吗？

基尔代尔：记得。那是一个计算一天中任意两个时刻之间的秒数或任意两个日期之间秒数的程序。这个程序现在还保留着，每次清理桌面时都能看到，就像衣柜里发现的一件旧衣服一样。



采访者：那你写的第一个专业程序是什么呢？

基尔代尔：是在我父亲办的一个航海学校里写的。以往我们都是手工为西雅图当地的一家出版公司准备潮汐表。后来我用FORTRAN语言写了一个计算潮汐的程序。那是为我赚钱的第一个程序，赚了大约500美元。

采访者：那你在CP/M操作系统上的工作又是如何开始的呢？

基尔代尔：那个操作系统实际上只是一个非常大的项目中的一小段。我当时用的是XPL，由斯坦福大学的比尔·麦克基曼（Bill McKeeman）编写、用于大型计算机的一种语言。我开发了一种类似的、用在微机上的编程语言，叫做PL/M。我要让PL/M驻留在8080微处理器中运行，这样就必须写一个接口与磁盘驱动器进行通信。结果证明那个操作系统，也就是被称作CP/M、用于微机控制程序的操作系统，也非常有用，这实在很幸运。

采访者：这么说，在开发CP/M的时候，你并没有想到它会如此成功？

基尔代尔：是的，我没想到CP/M会成为人们热议的焦点，但我却清楚地知道软盘会成为热点。我用了一年半的纸带。一个软盘驱动器要500美元，而一个带着复杂穿孔机的纸带阅读器却要2000美元还多。单单比较这两种驱动器的价格，就可以知道软盘将会在商业上取得成功。

采访者：有些程序员在发现代码中存在非常严重的问题时，会抛弃整个代码并重新开始。你这样做过吗？

基尔代尔：没有，因为我遇到的问题还从来没有严重到需要重新开始的。如果不能确认数据结构是正确的，我是决不会开始编码的。我每次废弃代码的时候，通常是因为底层的数据结构太糟糕，而不是因为所采用的算法。

采访者：在编码时，你会写注释吗？

基尔代尔：很少，除了在程序开头的地方写注释外，唯一加注释的地方就是数据结构。我不会对代码本身加注释，因为我觉得正确编写出来的代码本身就是很好的文档了。在写好算法后，我就开始直接在机器上写代码了，甚至不会先写在纸上，然后再输入到计算机中，我觉得好像没那个必要。实际的编码过程对我来说始终有点吓人，因为我不知道自己写的代码是否正确，也不知道接下去会怎么写。它似乎就那样冒了出来。有时候我意识到代码不是很正确，但直觉也会告诉我，它会涉及别的什么东西——代码最终写出来时会是对的，即使我在写代码的时候并不是很明确地了解。

神奇的地方在于，在某一时刻，整个程序一下子就成为一个整体了。这就像把一个逻辑的布尔表达式简化、再简化，突然间，就得到结果了。到了代码可以合并的时候，我可以肯定程序是可行的，而且毫无疑问，自己是以最佳方案编写程序的。我虽然不是完全理解整个过程，但它似乎对我来说确实是可行的，甚至在对数据结构和算法做出很大的改动时也是这样 。

采访者：编码是否一直都是一个未知和困难的过程？

基尔代尔：不是，当编码没有最后期限的压力时，我是很放松的。有时在乘坐飞机时，为了好玩，我会随身带上一个小的便携式打字机来编码。实际上，即便是有最后期限的压力，坐在终端前，让代码涌现出来也是很有趣的。这听起来很奇怪，但代码真的就这样从我的脑海中涌现出来，一旦开始编程，就不必想着该怎么编写了。

采访者：你曾遇到过那种情况吗，就是你的代码无法按照预想的方式工作？

基尔代尔：极少会有人在看了我的代码后说：“我们可以把它做得更好。”但有些时候，它就是没有办法按照期望的方式进展。在DR Logo解释器中的编



辑器就是这样一个例子。我知道有一些代码是不完全正确的——代码运行得还不错，但是编写得并不正确。后来接手的工程师集中精力来对付那段代码的编程，但是因为必须要交付产品了，你已经没有时间修改了。你希望这种事情永远不要发生，但有时它就是发生了，因此你要回去修复它，并就此学习一些编程风格的经验。

采访者：你觉得编程也可以练习吗，就像练琴一样？

基尔代尔：嗯，在某种意义上是可以练习的。计算机语言（标识语）的发明家西摩·佩伯特（Seymour Papert）认为孩子们可以通过摆弄齿轮等机械小玩意而变得具有创造性。通过这种玩耍而学习和练习的技能会转移到其他领域。佩伯特的观点正是我童年的经历。我父亲是一名技艺精湛的工匠。我常常待在旁边看他干活，一看就是好几个小时，然后跑到外面，拿起锤子和钉子模仿他。



作为编程基础的数据结构，从本质上看是机械的，就像我儿时玩的东西。所以，在这个意义上讲，我是可以练习编程的。最大的不同是，木制或铁制的东西需要花好几个小时来建造，如果做得不正确，必须回去重新做。而程序可以在瞬间完成修改。

采访者：作为程序员，还有其他积累本领的方式吗？

基尔代尔：你需要学习其他人的工作。他们解决问题的方法和他们使用的工具会让你以一个新的视角来审视自己的工作。在写程序前只需要学习一组为数不多的程序模块。例如，要写编译器，首先要写的是扫描功能，那是一个会用到很多次的小工具。一旦学会了这些工具，剩下的工作就只是把它们组合在一起。这里弄点儿、那里弄点儿，把这些功能模块都放到一起。查看其他人编写的程序可以为你提供构建条理清晰的代码的新思路。这就是为什么作为一个老师，我会花很多时间和学生们在一起，向他们展示我搜罗来的清晰算法模块的原因。

采访者：你已经谈了如何教别人。有没有什么人或什么事对你自己的编程风格产生过影响？

基尔代尔：我很讲求实效。我喜欢编写快速而小巧的程序，喜欢采用清晰而简洁的算法。我的这种编程风格来自早期的Burroughs 5500，一种当时非常

先进的计算机，这种计算机是以块结构语言ALGOL语言的理念为基础设计的。当时ALGOL语言编译器的代码可以算是最棒的。我花了很长时间修正和更改那个编译器。正是因为如此密切地使用ALGOL语言编译器，所以才影响了我对编程的想法，并对我的编程风格产生了深远的影响。幸运的是，ALGOL语言的理念成为了像Pascal和C这类流行编程语言的设计基础，所以那样的编程风格是适合我的。

采访者：曾经听说程序员们疯狂地长时间工作的事。你是怎样的？你有一定的惯例吗？

基尔代尔：我的步调在程序开发的各个阶段是不同的。在某些时候，代码如泉涌一般，所有的东西都同时出现在脑海中：所有的变量名，变量之间的相互联系，指针从哪里开始、在哪里结束，磁盘的访问等。各种各样的事情都浮现在脑海里，因为我不停地修改自己的想法，所以没有办法写在纸上。我花在设计上的时间比花在编码上的时间多，而且我从来没有在合理的时间内完成过一个项目。

当数据结构还在雏形时，需要高度集中注意力，让它们在脑海中成形。在这个阶段我通常会在早上3点开始工作，一直干到大约下午6点，然后吃晚饭，早一点上床睡觉，再很早地起床，不断推敲构思，直到数据结构定下来。

在平静的时候，我的工作节奏会放松一些，我会提出下一阶段的解决方案。我会有步骤地去解决问题，先把问题排好次序，然后一次一个步骤地去解决——步骤A，步骤B，然后步骤C。我试过了，除非把步骤B做完，否则就无法做步骤C。

我会间歇性地短期休假，因为我也很喜欢享受生活。这时我会外出，开飞机只是为了从工作中脱身出来。休假对工作很有帮助，因为我总会带回一些新的想法。

采访者：开飞机对你的编程有其他影响吗？

基尔代尔：我当然希望我的程序方案比我的飞行技术更出色。我听说有好几个程序员也是飞行员。我知道查尔斯·西蒙尼开直升飞机，弗雷德·吉本斯（Fred Gibbons）和维恩·瑞本（Vern Rayburn）也都对飞行很感兴趣。

程序员喜欢开飞机，因为开飞机是一个机械过程，就像编程一样。另外，



喜欢计算机的人也喜欢玩一些小玩意，而飞机上恰好装载了这些小玩意。所有你想要玩的，像仪表盘、轮子和旋钮，在飞机上都能找到。开飞机不是视频游戏，它是真实的，玩起来可能会有一些危险。计算机是很抽象的，而飞机却是真实的。

采访者：你可曾厌倦过编程吗？

基尔代尔：你是问我是否觉得工作乏味吗？我不觉得。在度假的时候，我期待着再次回到工作中。只有一种情况我不想回来工作，就是代码爆掉的时候。因为我处在压力之下，要让代码能够重新整合在一起是很难的。在所有的代码都被弄得凌乱不堪的时候，程序就像一辆被拆得散了架的汽车。所有零部件都摊在车库里，必须更换受损的部件，否则汽车永远都跑不起来了。这一点儿都不好玩，除非代码能再次回到最初的状态，否则真的是一点乐趣都没有。

采访者：你在工作中能找到美感吗？



基尔代尔：哦，当然有啦。当一个程序干净整洁、结构良好、前后一致时，它就是美丽的。我不能拿程序和蒙娜丽莎来比，但程序的简单和优雅确实是美的。不同程序的风格差异是耐人寻味的，很像是艺术评论家眼中达·芬奇的蒙娜丽莎和凡高的画作之间的差异。我特别喜欢LISP编程语言，它是如此赏心悦目。在LISP语言中有一个被称为M的简明表达式。用M表达式写的算法非常美，你几乎想把它装裱起来挂在墙上。

在做博士论文时，我曾试图解决一个很难的全局信息流分析（global flow analysis）问题。我知道肯定存在解决办法，但就是找不到答案。最后，我得到了一个清晰的数学模型，我用LISP对算法进行了编码。那个程序只用了两个小时就完成了，很美，精准地实现了我想要的功能。在当时，没有直接证据可以证明那个程序是能够运行的，但通过LISP运行的每个例子都按照预期的方式在执行。我还用XPL写了一个同样的程序，XPL是一个运行编译器的系统语言。后来，我证实了程序是正确的，我发现那个正确的程序是建立在那个非常漂亮的LISP程序概念上的，而不是建立在那个用相对丑陋的XPL程序开始的概念之上。

采访者：你认为编程是艺术还是科学？

基尔代尔：它当然有艺术的成分在里面。但很多的编程是发明和实现。这很

像一个木匠，在脑海里有他要做的柜子的样子。他要努力解决设计和施工的问题，直到柜子成型。我编程所做的差不多也是这样。

编程也有科学成分在里面，不过不是很多。实验科学意味着你要假设、尝试并比较结果，从这种意义上讲，编程也是科学。你可能有一个概念，知道检索系统应该如何工作，但是只有在使用足够的数据运行时，才能看到机制是可行的，并得到一些统计数据。

但是请记住，我是在一个特殊领域中编程的：编译器、操作系统、检索以及其他系统软件。而其他领域的，例如一个专业的图形程序员，可能对编程世界的看法完全不同。因图形程序员面对的更多是物质世界——比如说，他们谈论光源是如何影响物体的——他们在编程工作中可能会涉及更多的数学和科学。你知道吗，我认为编程对很多人来说也是一种宗教体验。

采访者：你说编程对很多人来说也是一种宗教体验，这是什么意思呢？

基尔代尔：嗯，如果你和一群使用相同编程语言的程序员谈论编程的话，他们几乎可以成为那种编程语言的传教士。他们组成了一个紧密结合的社群，坚持一定的信念，在编程时遵循一定的规则。就像是一个把编程语言当作是圣经的教会。



FORTH是一个很好的例子，它是一种编程语言，对于很多人来说，可能很接近宗教的体验。当FORTH刚出来的时候，其信徒就声称它做任何算法都能快上十倍。这是一个典型的宣言。如果你在这点或其他任何方面有异议，你会发现自己是对牛弹琴，而且你是绝对不会被容许加入“教会”的。我并不想贬低使用这一种语言的人。那是一个非常能给人帮助的团体，所用的也是一种非常有效的语言，但他们没有基于理性去讨论问题，他们是基于信仰的。说过这句话，我可能会收到一千封有关FORTH及人们对其宗教体验的邮件。但是就算我能整天鼓吹LISP的神奇，我也不会把自己归到一个特殊的派别中。

采访者：你认为计算机未来的角色将是什么？

基尔代尔：大致说来，我们的技术会简化机械的过程。这就是为什么计算机会如此成功的原因，我们先是用真空管、然后又用半导体完成了通常由齿轮、车轮和继电器所做的事情。比如说，看看汽车制造业。在汽车制造中，越来越多的过程都正在转向半导体或其同类产品，像1984年的Corvette就是这样。

在半导体取代了速度计和转速表的电缆后，汽车变成了一种更便宜、更可靠、更容易生产的产品。计算机系统正在经历着相同的变化。现在的硬盘驱动器是一个机械装置，因为是机械的，所以我们知道它最终肯定会消失。我们不知道它将如何消失，但知道那是一个主要目标。

一些小工具和加工方法很难用半导体来制作，比如汽车上的车轮轴承，它们会继续以机械的形式存在。但日常生活中很多其他东西都会从机械化过渡到电子化。印刷行业就是一个很好的例子，现在CD-ROM和光存储变得很重要了。计算机有助于印刷业脱离机械化的过程：操作印刷机、手工排版和粘贴、设置相机，半导体将取代机械化的过程。但计算机却不会就此止步。现在计算机控制了印刷业的生产，但没有控制信息的实际展示。

现在，一个非常大的瓶颈——也是个人计算机行业不景气的一个原因——就是，我们想不出在实现了电子表格和文字处理后，还能用计算机做些什么。我们不知道下一步是什么。我们被难住了。



这又回到了我刚才谈过的，编程依赖的是信念而不是理性。最终的问题是，我们这个团体把我们所理解的大型计算机中的基础结构、语言和概念应用到了微型计算机的发展上。当我们趋向于把计算机当作控制器来使用时，会发现处理器之间的交流将变得比它们执行的过程更重要。那时我们将不得不改变编码方式。这将是一个非常缓慢的演进过程。

采访者：*所以未来的发展实际上取决于我们是否有能力从旧的思维模式中摆脱出来？*

基尔代尔：在微处理器刚出来的时候，我强烈地感到，它们应该主要用于嵌入式处理器，相互通信，协调从机械化过渡到电子化的发展过程。我觉得这是计算机行业的发展方向。我认为这些小处理器替代的是随机逻辑，主要的用户是工程师。实际上，劳伦斯·利弗莫尔实验室中有人建议我开发微机上的BASIC——那大概是在1974年。我告诉他这是我听过的最愚蠢的想法。这些微处理器将放到库存控制系统、阴极射线管显示器和字处理器之类的工具中，谁愿意为这些微处理器开发BASIC呢？显然，在这点上我错了。事实证明，我的一个学生，戈登·欧班克斯（Gordon Eubanks），我辅导他做的论文，他的C BASIC就做得非常好，就跟保罗·艾伦和比尔·盖茨做得一样好。

如果想推动计算机的进步，就必须以某种方法挣脱对微型计算机的思维

方式。了解情况的人不会想着再买另一套计算机系统了。他们已经买了一套系统，却没能真正用起来。他们不想再被骗一次了。我们说的是95%的计算机用户，而不是那余下的5%。电视机每年的销量是1600万台，安装了嵌入式微处理器的小装置为什么不能同样卖到1600万台呢？

采访者：你刚刚提到了CD光盘及其对印刷业的潜在影响。在计算机的演变中它会发挥其他作用吗？

基尔代尔：显然，光存储将把计算机行业带向一个新方向。在使用软盘时，我们只是在大型计算机的基础上制造了小型计算机。直到今天，我们也并没有能走多远。

光存储就完全不同了。我们不再谈论计算了，我们谈论的是为人们提供信息。现在人们购买个人计算机可能是因为别人告诉他们应该买一台，但有了光存储，人们去买计算机是因为他们想得到信息。计算机行业与出版业将会有更多的竞争。



采访者：这么说，信息可以采取电子百科全书的形式，就像你现在光盘上做的？你是如何构想那些功能设计的——包括检索系统和增强功能？

基尔代尔：我从最初的产品概念得到想要做的产品的总体思路，并从核心点开始编码，让程序自然扩展。只要我不限制基本的数据结构，就可以一直添加功能。我们做了一个叫做“知识光盘”的视频光盘，很好地转换到了光盘的检索系统中。所有的文字都采用以一组二维像素信息表示的点阵字体。使用像素的一个非常好的额外作用是，图像可以又好又简洁地融入到整个系统中。这样在向光盘上已经存在的文字中添加图像时，就不必回头把所有的设计都重做一次。

如果一个设计不允许今后添加新功能，那是有问题的。如果必须重写程序，花费的时间可能会让你失去竞争优势。程序的灵活性显示了一名好的程序设计师与一个只是写写代码的程序员之间会有多么大的差异。

现在，我们全力地宣传它，想让尽可能多的人知道。我们希望这个产品的第一部分能在1986年准备就绪。从经济的角度看，我们别无选择，只能尽可能快地使用这个技术。这个技术是最好的。然后我们就要确保对这项技术有许可权。

采访者：知识系统是你认为的家用市场发展方向的一部分吗？

基尔代尔：是的。人们通常不在家里办公。人们在家里做的有些事情是与工作相关的，比如记录纳税情况，或是运作一个小型的家庭企业。但多数人回家就是为了休息。我认为游戏和娱乐是有价值的。我们很难搞清楚如何给人们在家中带来娱乐。现在电视在这方面做得很好，与电视剧“王朝”竞争是非常困难的。

计算机应用软件还可能用来帮助孩子们学习。我14岁的女儿正在上一些很难学的课程，需要有人帮助她来学习。如果有这样的计算机应用软件，就能为我带来直接的好处：孩子学习好，对她的未来就有益。这显然是计算机发展的一个重要领域。

计算机发展的另一个领域是为所选定的科目提供相关的大众信息，例如医学。人们去看病的原因有很多，有些是心理上的，但有时只是想了解医疗信息。看病的费用很高，如果有便宜的方法提供这些信息，人们是愿意接受的。再举一个例子：如果要买汽车，我会到洛杉矶、旧金山、圣何塞各处的车行中逛逛，为的是能获得最便宜的价格。但你几乎无法得到汽车经销商的真实信息，因为他们若不想让你以这种方式来买车，就会把信息封锁起来的。如果有个系统能够提供这些信息，我一定会用的，因为它可以帮我少花很多钱，更不用说可以节省多少时间了。我会愿意支付合理的费用来得到它。



我们要开发的应用程序是这样的：如果人们买了这些程序，就会给他们带来明确的经济效益。这就是为什么我们做在光盘上的第一个应用程序是百科全书的原因。大家都知道百科全书通常要花费大约1000美元。如果与印刷的百科全书在相同的价格范围内，那人们宁愿去买一台带百科全书的计算机。

采访者：这种计算机的友好性会怎么样？

基尔代尔：嗯，我认为与友好性无关，因为如果最终要用程序去做什么有价值的东西，它可能是比较复杂的。

有些人提议说，如果可以用一种自然语言来输入，计算机可能会更友好。但是自然语言可能是最糟糕的输入方式了，因为它可能会相当含糊。从计算机上获得反馈信息的过程会很耗时，你宁可把这些时间花在专家那里，直接从他们那获得所需的信息。

专家系统最终将是一个用户友好的系统。但是距离真的拥有专家系统，我们还有一段很长的路要走。医生专家系统虽然是一个非凡的产品，但却极为复杂。它必须得是完美的。总有一天，我们会有那样的专家系统。尽管我不知道这离我们还有多远，中间也会有很多问题需要解决，但这部分工作却会很有趣。

续写传奇人生

时任Activenture公司总裁，同时担任Digital Research公司董事会主席。Activenture是最早为CD-ROM开发应用软件的公司，致力于探索光盘出版的潜力。但基尔代尔最终还是卖掉了该公司的大部分股份。1991年，Digital Research亦被出售给Novell公司。此后，富有的基尔代尔举家离开硅谷，搬到德克萨斯州奥斯汀市郊的西湖山群（West Lake Hills），他在当地拥有一座湖边别墅，并时常作为志愿者参与对儿童艾滋病患者的援助工作。



基尔代尔从1983年开始，连续6年参与主持PBS电视台的“计算机编年史”（Computer Chronicles）节目。他言谈举止温文尔雅，颇具个人魅力，格外受人欢迎。

1992年，华盛顿大学邀请基尔代尔以杰出毕业生的身份，参加计算机科学系系庆活动，但他对学校邀请“哈佛肄业生”盖茨发表主题演讲非常不满。作为回应，他开始写作回忆录*Computer Connections*，这本回忆录并未公开发行。在这本回忆录里，他对人们不重视软件的优雅而感到失望，并指责盖茨从他和这个行业里拿走了太多，甚至在附录中干脆声称DOS为“直接简单的盗窃”。此外，他还指责IBM故意用不同的价格销售PC-DOS和CP/M-86，使得CP/M被边缘化。

1994年7月11日，基尔代尔因头部严重受伤，在加州蒙特雷去世，享年52岁。

```
BASIC MCS 8080/Z80       GATES/ALLEN/DAVIDOFF       MACRO 55B(272)-1 21:12 21-FEB-78 PAGE 1
F3      MAC      17-FEB-78 19:16                VERSION 4.00 -- NOT MANY FEATURES TO GO

COPYRIGHT 1975 BY BILL GATES AND PAUL ALLEN
-------------------------------------------

ORIGINALLY WRITTEN ON THE PDP-10 FROM
FEBRUARY 9 TO  APRIL 9

BILL GATES WROTE A LOT OF STUFF.
PAUL ALLEN WROTE OTHER STUFF.
MONTE DAVIDOFF WROTE THE MATH PACKAGE.

THINGS TO DO:
PRINT PUNCTUATION MANDATORY
MULTIPLE LET
MAKE STACK BOUNDARY STUFF EXACT
*
RADIX 10                              ;THROUGHOUT THE NON-MATH PACKAGE ROUTINES
.P==0                                 ;FOR SIMULATOR FIXUPS
NUMLEV==19+LENGTH+5                   ;NUMBER OF STACK LEVELS RESERVED
IFN     NCRBAS,<NUMLEV==NUMLEV+4;>
                                      ;WITH AN EXPLICIT CALL TO GETSTK
IFE     TIMSW,<
IFE     LPT3,<
IFE     NCRLPT,<
LPTLEN==80>                           ;WIDTH OF LINE-PRINTER
IFN     NCRLPT,<
IFE     CPMIONC,<
        EXTERN  LPTLEN,NLPPOS,FINLPT>>
IFN     [illegible]
        LPTLEN==132>
>                                     ;END LPT3 COND
        BUFLEN==72                    ;INPUT BUFFER SIZE
IFN     MSD,<
        BUFLEN==[illegible]>          ;MAKE BUFFER LARGER FOR MSD
IFN     LEN2,<BUFLEN==255>            ;LONG LINES
>                                     ;7/5 BASIC
        STRSIZ==4
IFN     LENGTH&2,<STRSIZ==3>
        NUMTMP==3                     ;NUMBER OF STRING TEMPORARIES
IFN     LENGTH&2,<NUMTMP==10>
IFE     TIMSHR,<
        CONTOU==15                    ;CHARACTER TO SUPRESS OUTPUT
        LISTEN==0                     ;ON MEANS LISTEN FOR ^C
IFN     REALIO,<
        LISTEN==1>
>                                     ;7/5 BASIC
        BUFOFS==0
IFN     LEN2,<
        BUFOFS==2>                    ;MUST CRUNCH INTO EARLIER PLACE FOR SINGLE QUOTE
IFE     TIMSHR,<
IFN     FAST,<
KBFLEN==BUFLEN+<BUFLEN/4>>            ;MAKE KRUNCH BUFFER SOMEWHAT
                                      ;LARGER THAN SOURCE BUFFER (BUF)
>                                     ;7/5 BASIC

BASIC MCS 8080/Z80       GATES/ALLEN/DAVIDOFF       MACRO 55B(272)-1 21:12 21-FEB-78 PAGE 1-4
F3      MAC      17-FEB-78 19:16                SOME EXPLANATION

SUBTTL  SOME EXPLANATION

COMMENT *

ALTAIR BASIC CONFIGURES MEMORY AS FOLLOWS:

LOW LOCATIONS   (NON-TIMESHARING VERSION)

        RST     SUBROUTINES

                0       STARTUP
                        INITIALLY A JMP TO THE INITIALIZATION CODE
                        BUT CHANGED TO A JMP TO READY.
                        RESTARTING THE MACHINE AT 0 DURING PROGRAM
                        EXECUTION CAN LEAVE THINGS MESSED UP.

                1       SYNCHK
                        A CHECK IS MADE TO MAKE SURE THE
                        CHARACTER POINTER POINTS AT A SPECIFIC
                        CHARACTER. IF NOT THE "SYNTAX ERROR"
                        ROUTINE IS CALLED. IF SO,
                        THE CHRGET RST IS DROPPED INTO SO
                        THE CHARACTER AFTER THE MATCHED
                        ONE WILL BE PUT IN [A] AND
                        THE CONDITION CODES WILL REFLECT THIS
                        EXAMPLE: SYNCHK THENTK (THE MATCH CHARACTER IS
                        GIVEN IN THE LOCATION AFTER THE RST)
                        WOULD CHECK TO MAKE SURE [H,L] POINTED TO A THENTK
                        AND IF SO FETCH THE NEXT CHARACTER INTO [A].
                        IF NOT, A "SYNTAX ERROR" WOULD BE GIVEN.

                2       CHRGET
                        USING [H,L] AS THE TEXT POINTER
                        THE TEXT POINTER IS INCREMENTED
                        AND THE NEXT CHARACTER IS FETCHED INTO [A]
                        IF THE CHARACTER IS A " " IT IS SKIPPED
                        OVER AND THE NEXT CHARACTER IS FETCHED.
                        THE STATEMENT TERMINATORS ":" AND 0
                        LEAVE THE ZERO FLAG SET.
                        THE NUMERICS "0" THROUGH "9" LEAVE THE CARRY
                        FLAG SET. THE CURRENT CHARACTER CAN BE
                        REFETCHED INTO [A] BY DOING A MOV A,M.
                        IF THE CONDITION CODES MUST BE SET UP AGAIN
                        DCX H,CHRGET WILL WORK. IT IS VERY DIFFICULT
                        TO REEXAMINE THE CHARACTER BEFORE THE CURRENT
                        ONE SINCE SPACES MAY BE IN-BETWEEN.
                        DCX H,DCX H,CHRGET WILL NOT ALWAYS WORK.

                3       OUTCHR
                        THE CHARACTER IN [A] IS PRINTED ON
                        THE USER'S TERMINAL. [A] AND THE
                        CONDITION CODES ARE PRESERVED.

                4       COMPAR
                        [D,E] AND [H,L] ARE COMPARED AS UNSIGNED
```

从微机的第一个高级语言BASIC 8080的代码清单上摘录的一段。注意第172至174行的著作人信息，以及第367行提到的MITS Altair计算机信息。有关此程序和Altair上的其他程序，请参考附录中更多的摘录。



5

比尔·盖茨



作为微软的CEO，威廉·H. 比尔·盖茨（William H. Bill Gates）被认为是当今个人计算领域和办公自动化行业一个强有力的推动者。比尔·盖茨从年轻时就开始了计算机软件的职业生涯。当盖茨和微软的联合创始人保罗·艾伦还在华盛顿州的西雅图上高中时，两人就开始做起了编程顾问的工作。1974年，盖茨在哈佛大学读本科时，他与艾伦合作为第一台商用微型计算机MITS Altair开发了一套BASIC编程语言。在那个项目顺利完成后，两人创办了微软公司，为新兴的微机市场开发并销售软件。

微软为软件产业在编程语言、操作系统和应用软件等各方面设定了标准。盖茨为微软提出了新产品的创意和技术发展的远景。在开发新产品时，他还会亲自指导技术小组，投入时间复审和完善微软所销售的软件。

盖茨出生于1955年，是西雅图本地人，至今一直居住在那里。

* * *

采访者：显然，作为微软的CEO，你的责任重大。你现在仍在编程吗？

盖茨：我现在不编程了。我仍会在算法设计和基本方法上提供帮助，有时也会看看代码。但自从完成IBM PC BASIC和Model 100上的工作后，我就再也没机会自己动手编写程序了。

采访者：在微软的软件开发过程中，你扮演了什么样的角色？

盖茨：我做两件关键的事情。一是选择在程序中放入哪些功能。为了做到这一点，必须合理把握什么事情容易做、什么事情不容易做。还必须明白你追求的产品系列的策略是什么样的，并要关注硬件领域的进展。

此外，我还致力于实现新功能的最佳方案，也就是如何把新功能做得既小又快。例如，我写过一个备忘录，是关于如何设计和实施Excel中的一项功能的：每当屏幕发生变化时，程序都要重新计算其中的公式。

在公司成立后最初的4年，没有一个微软的程序是我没有参与编写和设计的。在所有这些最初的产品中，无论是BASIC、FORTRAN、BASIC 6800还是BASIC 6502，没有一行代码是我没有检查过的。但现在我们有大约160名程序员了，所以我主要是做产品和算法的复查。

采访者：你认为你在编程上最大的成就是什么？



盖茨：那得说是为8080编写的BASIC了，因为程序当时所产生的影响，而且因为我们设法把程序做得很小巧，非常适合当时的使用场景。那是我们决定创办微软时所编写的最早的程序。

我们三个人都清楚地记得那个最早的程序。我们得到了一个机会，在新墨西哥州的阿尔伯克基花了整整一个夏天，把程序彻底重写了一遍。我认为可以节省几个字节，让程序更精简。我们非常非常仔细地调试着，最终得到了那个4K的BASIC解释程序。

当你非常了解一个程序，觉得没有人在看了程序后会说“还可以做得更好”时，那种感觉真是太棒了，而且程序用在了很多机器中，让人觉得编写那样一个程序是件很兴奋的事情。

我还非常喜欢为Model 100编写的那个程序，特别是我们把一个非常有用的小编辑器压缩到了软件中。我和一个名叫杰米·玲木（Jey Suzuki）的日本程序员合作完成了那项工作。我们在非常有限的时间内完成了那个项目。如果编写的软件要烧入ROM，你是没有机会可以犯错的。

采访者：你认为计算机编程中最困难的部分是什么？

盖茨：最困难的部分是确定采用什么算法，然后还要尽可能地简化算法。做到最简单的形式是很难的。必须在心中模拟程序是如何工作的，必须完全了解程序各部分是如何一起工作的。最好的软件是其中有一个程序员完全了解

程序的工作方式。要做到这一点，必须要特别热爱编程，集中精力让程序变得极为简洁。

采访者：随着计算机能力越来越强大、内存越来越多，编程会变得越来越复杂，还是会变得越来越拙劣呢？这对人们编写程序的方式会产生什么样的影响？

盖茨：我们已经不再生活在每一个程序都精雕细琢的时代了。但是你会发现，程序要做到顶尖，最重要的是，那些关键的内部代码都是由少数几个知道自己在做些什么的人编写出来的。

现在把程序压缩到4K的内存区域中已经不那么重要了。你在很多情况下会看到人们使用C语言，而不再使用汇编语言了。遗憾的是，很多程序都太大了，已经没有一个人可以真正了解整个程序的所有部分了，所以能得到的共享代码也不是很多。此外，因为一直要在同一个程序中添加新功能，所以也没有太多机会让你回去重写代码。

最糟糕的程序是原来的程序员在开始时没有打好基础，而他们也没有再参与到程序的后续开发中。在这类程序上继续工作就会遇到一种我所说的"实验性程序"的情况。程序员对那些程序了解得太少，他们不知道改动之后会影响什么，比如说会不会影响运行速度。他们可能会使用已经存在的代码，他们也有可能并不知道如果修改了代码，会破坏何种依赖关系。于是他们加入了新代码，并在运行之后说："噢，看哪，它不是那样运行的。"这种处理程序的方法效率非常非常低，但很多项目到了最后却都是这样的。

采访者：在一个像微软这样有160名程序员的公司中，你是如何创造一个环境以确保能开发出成功软件的？

盖茨：一种方法是建立小型的项目团队，通常是四五个人一组，其中一个人经证实有能力掌控整个程序。如果这个项目带头人遇到不确定的事情，他会与经验更为丰富的程序员一起讨论。

我们的部分策略是让所有程序员在进入编码阶段之前都先想清楚每一件事情。编写程序设计文档是至关重要的，因为在把问题当做算法看的时候，问题会得到很大的简化。可以说算法是最简单的形式，从中可以看出问题在什么地方是重叠的。

另一个重要因素是代码复查，要确保代码是看过的，看看资深人士是否



能提出如何做得更好的建议。而且你必须参考类似的、做得特别特别好的项目。程序员可以看看以前其他人是怎么做的，从其他项目获得改进自己程序的想法和经验。

采访者：那些程序的构思是怎么来的？

盖茨：嗯，说实在的，还真没有什么正式的流程。在微软，通常在晚上或周末会有一个集思会。大家会有些大致的想法，比如说，我们要做世界上最好的字处理器，我们希望技术出版部门借助这个字处理器能够做他们想做的每一件事。为此我们会坐下来讨论：怎样才能让程序真正快捷？能够嵌入绘图功能吗？能够让字体平滑但又不降低程序的效率吗？各种各样的问题都会透彻地讨论，接着就会出现一些好的想法。

采访者：大体说来，程序的构思是集体智慧的结晶？

盖茨：对于决定要开发哪些程序，我们有一个相当大的团队来提出建议。然后会有一个筛选的过程，最后由我决定哪些想法是有意义的。我要确保项目有一些项目带头人，能够亲自参与项目，确保产品开发成功。为了开发一个产品并制定新的世界级的标准，需要投入非常大量的资源，所以我们选择的项目会非常非常少。



采访者：很多人都在谈论大型软件公司要想吸引能开发出优秀软件的人才有多难，因为这些大侠都太特立独行了，他们喜欢独自作战。在微软，你们是如何吸引并留住那些优秀人才的？

盖茨：优秀的程序员对于软件产品的开发是至关重要的。但是我们不赞同独行侠的做法，不会仅仅因为一个人很优秀，就允许他在代码中不添加注释，或允许他不与其他人沟通，或是允许他把自己的想法强加给别人。

我们希望程序员能够相互尊重。我认为大多数优秀的程序员都希望周围有其他优秀的程序员。当他们想出了一个很好的算法时，他们希望周围有能够欣赏其绝妙之处的同事。因为你在构想那个算法、脑海中产生那样一个模型时，那是个寂寞的事情。如果你原来以为处理过程很复杂，但却找到一个办法，让过程变得很简单，那种感觉好极了。不过你需要从其他程序员那里得到一些反馈。如果已经有了几个优秀的程序员，就会吸引更多优秀的程序员。

传统的管理规则是，程序员的管理者总是一个更加出色的程序员，没有

我们所说的“技术倒挂”，让程序员为一个不知道编程为何物的人工作。我们仍旧遵循这一理念：在一定的级别上，我们会用业务经理，但不会用非程序员管理正在开发的软件项目。

采访者：你认为开发优秀的程序有什么特定规则吗？

盖茨：有些人刚一进到项目中就开始坐下来编码，而有些人则在编码之前把所有的过程都想清楚，我认为你会发现那些一开始就坐下来编码的程序员只是在把那些代码当做草稿使用。那些在他们头脑中思考的内容才是最重要的。

你必须得有非常聪明的程序员。一个优秀的程序员会一直不断地思考所开发的程序，无论是开车还是吃饭。不停地思考问题，需要耗费大量的脑力。

采访者：你的编程风格是什么？

盖茨：我喜欢在坐下来编写代码之前先把整个设计方案构想清楚。而在完成代码后，我喜欢回去把它从头到尾再全部重写一遍。

编写程序最重要的部分是设计数据结构。接下来重要的部分是分解各种代码块。在开发到那一步并写出代码之前，你无法敏锐地感知那些公共子例程应该是什么样的。

我所写过的真正优秀的程序都是在开始动手编程前已经花了大量时间去思考的。我在高中时为一台小型机编写了一个BASIC解释程序。我在那个程序中犯了很多错误，后来我得到机会，看到了一些其他的BASIC解释程序。这样当我在1975年坐下来编写微软的BASIC解释程序时，问题已经不在于是否能写出程序来，而在于能不能把程序压缩到4K，并得到非常快的运行速度。整个过程中我都在紧张地思考：“速度够快了吗？其他人会做得更快吗？”

我一直记得一个人，他是我在TRW公司遇到的，名叫诺顿。我没有做到特别好的时候，他总会给我指出来。所以当我草率或偷懒的时候，我总是想象着他会走过来，看一看程序，然后告诉我：“你看，这儿有一个更好的方法。”在编程的过程中很容易引入一些小的低效率的做法，要想获得良好的感觉，就必须时刻保持警觉，不能让这类东西侵入，这也是为什么有时候和其他人在项目中一起工作会让你觉得痛苦的原因。因为他们从来都不能按照你希望的方式编写代码。记得当我们在做BASIC解释程序时，我常常回去重新编写其他人的部分程序，但又没有做出什么特别大的改进。这种做法会



困扰你的同事，但有时你觉得必须那样做。

采访者：*在和团队一起工作时，你总是设计方面的带头人吗？*

盖茨：是的，在我直接参与的所有项目中，我都是设计带头人。最初编写BASIC时，我在纸上画出了设计草案。我的合作者保罗·艾伦设计并实现了所有开发工具。

在坐下来编码之前，大部分指令都已经在我的脑海中运行了。这并不是说所有的设计都已经很好地完成了，我也会做修改的，但所有好的想法在开始编码之前就都想到了。如果在某处有一个bug，我会觉得很不舒服，因为如果存在bug，说明你在脑海中的模拟是不完美的。而一旦脑海中的模拟不完美，在程序中就有可能会有几千个bug。我真的很讨厌看到一些人在编程的时候不动脑筋。

我最有趣的编程经验是在我们编写BASIC程序的时候。那时我已经开发完成了8080上的BASIC程序，接下来有大约两周的时间可以和马克·查姆伯林（Mark Chamberlain）一起开发6809版本的BASIC程序。在那两星期的开始几天，我读了新的指令系统，接着写了三四个程序。我还读了一些其他程序，看看别人是如何使用指令集的。把自己已经理解的问题映射到这个新的指令集中，看看如何将它们紧密地结合在一起，真是非常有趣。



如今的程序变得非常臃肿，因为人们在程序中加入了特殊检查，所以功能的提升往往会降低程序的效率。当他们想增加功能时，就会加入某类检查，却不考虑这样做也许会降低程序效率。必须有一个程序员能完全了解整个程序，防止这种现象的发生。比如我们在完成BASIC程序后，我和其他最初的开发人员都离开了那个项目，程序在此后大约3年的时间里都没有开发过任何创新的东西。直到在过去的一年半中，我们才有程序员感到自己完全掌握并全面了解了那个BASIC程序，能够说："哦，放些子例程进去，很容易就能去掉行号了。"直到那时我们才对程序做了更新。其实我们一直都想更新那个程序，但是如果程序员仅仅会在表面打些补丁，而不知道如何深入部件或语句分析器的内部，你是没有信心去动那些程序的。

的确，我们会允许程序比本该有的稍臃肿些。但在速度方面，不允许因懒惰而不设法使程序尽可能地快，因为用户会留意到程序是否非常非常快，即使他们未必能直言不讳。那些最成功的软件，程序的运行速度都非常快。

采访者：你是如何在速度和性能之间进行权衡的？

盖茨：有时候是需要在添加功能和快速运行之间进行权衡的，但也有其他办法，即使增加很多很多功能，速度也照样快。一般来说，你要确定程序中最常见的情况是什么，并要确保它们运行通畅，不会陷入到那些特殊情况的检查中。因为如果在主要的交互循环流程中有各种各样的检查，程序就会比别人的慢。

采访者：当你提出一个想法，要做世界上最好的字处理器时，你们是怎样做的，如何设计的？是否考查过市面上所有的字处理器？

盖茨：是的，我们考查了其他字处理器的功能。在考查时你会想："会有人在屏幕上对字体做平衡处理吗？还是说屏幕上显示的与打印出来的完全一样？运行速度怎么样？"通常，那些尖端的产品中，有人会采用非常非常昂贵的硬件，使用蛮干而不动脑筋的方式解决问题。我们不能那样做，因为运行我们软件的微机速度有限。我们要做的很多事情在高端计算机上都已经有了，我们要在成千上万的微机上也实现这些功能。

你可以在产品中使用大量的技巧。在建立功能列表的同时你要尝试回答下面这个问题："为什么我们的算法比别人的都好？"功能在某种程度上也可能使产品变得很糟，因为功能越多，用户手册就越厚。而功能只对那些肯花时间去使用它们的人才有用，这不像速度——如果打印页面的速度更快，在屏幕上显示的速度更快，重新计算的速度更快——那么产生的价值是惊人的。如果能够提供一些简单的命令，程序通过这几个简单的命令就能有效地满足用户的需求，那么结果会好得多。非常好的程序的一个标志是在内部也能遵循简单的理念。如果程序想实现复杂的功能，可以通过简单的内部操作调用代码，而不必从头开始去做一系列复杂的操作。

采访者：最终用户的重要程度如何？你如何知道数据库管理员对数据库或报表的真正需求和要求是什么？

盖茨：嗯，对于最终用户究竟想要什么，有些程序员用不着假装对此有很直观的洞察力，但他们仍是世界级的程序员。不过对市场的认识是很重要的，尤其是在应用程序组，所以我们有全职的工作人员，专门向客户演示代码、调研行业规范或其他相关信息。在微软刚成立的时候，我们只开发系统程序。我们是程序员，所以知道程序员想要的是什么。因此我们编写了BASIC语言。



采访者：BASIC在哪些方面是最具创新的？

盖茨：我们提供了允许程序员使用机器全部能力的方法。我们嵌入了PEEK和POKE指令，程序员可以读写机器状态。我们提供了名为TRON和TROFE的跟踪调试例程。即使是高级语言，用户也可以把他们想做的各种古怪的小功能添加到计算机中，并且无需使用BASIC程序就可以了解内存使用情况。我们让他们觉得计算机是由他们自己掌控的。

为了把BASIC放到4K的内存中，我们使用了叫做单一表示法解释程序的机制。这是一个非常好的选择。此前我从来没有见过哪一个解释程序这样做过。虽然有些冒险，但我却对这种机制信心十足。我在脑子里把它过了一遍，感觉很好。

采访者：你在编写BASIC解释程序的时候，可曾想过它会如此成功吗？

盖茨：不，根本没有想到。当时保罗·艾伦给我带来一本介绍Altair的杂志，我们就想："老天，我们最好在上面做些事情，这些机器肯定会很受欢迎的。"于是我不再去上课了，我们开始没日没夜地工作起来。程序的雏形花了大约三个半星期，然后花了大约8个星期让程序最后按我真正喜欢的方式充分完善。过了些时候，我又回去把它重写了一遍。



没有哪一个优秀的程序员会坐在那里说："我要赚一大笔钱。"或者说："我要卖出成百上千个拷贝。"因为这种想法不会对你解决问题带来任何帮助。一个优秀的程序员想的是：是否应当把整个子例程都重写一遍，这样就能够让4个人而不仅仅是3个人调用了？能让程序快10%吗？是否应当好好想一想这里的常规状况是什么，这样就可以对校验条件进行排序了？如果是一个优秀的程序员，你会让所有例程互相依赖，这样就不会有什么致命的错误了。这就是为什么必须有精准的判断，愿意备份并修改程序的原因。

采访者：当多人参与到同一程序中时，怎样确保各种不同的人都能融洽地一起工作呢？

盖茨：嗯，首先，项目组必须由能够互相尊重的成员组成，因为这个工作需要密切配合，就像是打一场比赛一样。编程项目需要很多的判断力和创造力。有些优秀的程序员无法融入团队工作，他们喜欢干自己的。但我觉得优秀的一个要素来源于学习如何与其他人一起工作，并教导别人。我会因为和我一起工作的人成长为优秀的程序员而感到欣慰，虽然不像我自己写程序那么开

心，但也是很好的事情。我让别人成为一名优秀程序员的方式就是坐下来和他详谈，给他看我写的代码。在一个项目团队中，你的代码也是大家的代码。

采访者：这种过程是自然演变的，还是刻意实施的结果？

盖茨：在我和保罗创办微软之前，我们参与过一些大型的软件开发项目，那些项目可真是灾难。他们只是在不停地往项目组加人，却没有人知道要如何去稳定项目。我们发誓说自己可不能做成那样，我们要做得更好。要多花一些时间组织规划开发组的工作，这种想法对我们一直都非常重要。

最好的方法是显而易见的：保持项目团队精简；确保小组中每个程序员都非常聪明；为他们提供强大的工具；有一套公用术语，以便大家很有效地沟通。而在这些小组外，找一些经验丰富的资深人士，在遇到问题时能够出谋划策。在项目过程中遇到的困难类型有着惊人的共性。在做设计评审时，我非常乐于根据我自己的项目经验提出建议。

采访者：你认为人们编程或操作计算机的方式会发生很大的改变吗？



盖茨：软件开发工具变得比以前好多了。最终我们可能只需要取得规格说明，以及一个怎样有效使用机器的说明，然后就会有一些超高级的编译器来完成很多目前由程序员所做的工作了。

尽管一个编译器，比如C编译器，还无法像人编写的代码那样好，人们仍旧会因此而感到非常满意。但在今后三四年中，我们可以把这个过程中相当多的部分都做到机械化。人们仍然会设计算法，但是很多实现工作都可以通过机器完成了。我认为，在未来5年内，出现的工具所能完成的工作将会像程序员一样好。

采访者：你刚才提到了数学。计算机科学和数学之间的关系是什么？

盖茨：数学对计算机科学有着很大的影响。大多数优秀的程序员都有一定的数学背景，因为它有助于学习证明定理过程中的纯正性，在证明定理的时候不能做模糊的陈述，只能做精准的陈述。在数学中，不仅要建立完整的特征描述，而且要以很不明显的方式把定理结合起来。你常常会去证明一个问题是可以在更短的时间内解决的。数学与编程有很直接的联系，因为我是这么看待这个问题的，所以我的这种观点可能比别人更强烈一些。我认为两者之间有着天然的联系。

采访者：计算机科学真的是科学吗？

盖茨：会是的。这是一个非常新的事物。现在我们希望程序员能完成的一部分工作，在以前是人们用来做博士论文时要做的工作。计算机科学正在飞速发展，但它不像数学，数学天才在300年来一直在不断地丰富着数学理论，而人们愿意投身到计算机领域却不过是最近20年的事情。一些卓越的人加入到计算机科学领域并做出了贡献。和以前相比，编程现在已经是非常主流的工作了。人们在很小的时候就开始接触计算机，这有助于改变计算机科学领域的思维方式。很多优秀的程序员在他们十几岁时就开始编程了，在那个年纪思考问题的方法也许会更灵活一些。

在过去，人们认为单单成为优秀程序员是不够的，你还得去管理别人或处理其他事情。幸运的是，这种情况正在改变，现在人们认识到计算机是一门科学，是值得坚持下去并教授给其他人的。

采访者：经过多年的经验积累后，编程是否一定会更容易呢？



盖茨：不，我认为在过了最初的三四年后，就会非常明显地显现出你是否是一个优秀的程序员。刚开始的几年中，你可能会更多地知道怎样去管理大型项目和不同个性的人，但在三四年后，就能很清楚地看出你会成为什么样的程序员了。在微软没有哪个程序员是在平庸了几年之后突然间一鸣惊人的。我和一个人谈谈他的程序，马上就能知道他是否是个好程序员。如果他真的很棒，每个细节他都会脱口而出。

就像下棋的人一样。如果你特别喜欢下棋，会很容易记住10盘棋中的每一步，因为你已置身其中了。其他人看到国际象棋选手或程序员能记住每个细节时，觉得他们像个怪物。其实这很正常。即使到了今天，在我写了微软的BASIC程序10年后，我仍可以在黑板上大段大段写出当时的源代码。

采访者：你在编程时是什么样的感觉？

盖茨：在编译程序并计算出正确的结果时，我会非常开心。这是真的，不是开玩笑，所有伟大的事情都会有感情因素，编程也不例外。人们很想一上来就开始编码，但是如果编写代码只是为了得出结果，过后却发现所有难题都还没有处理，那么没有什么比这更糟糕的做法了。因为如果真是那样的话，你就不得不修改已经完成的程序。为了享受编码并看着代码运行起来的乐趣，我喜欢等一等，把基础打好。这就像吃饭时把最好的食物留到最后才吃一样。

采访者：你看到年轻的程序员和老程序员的编程方式有什么不同之处吗？

盖茨：今天刚刚起步的程序员根本用不着对程序进行压缩，他们认为要用到的资源总是能马上得到，所以让他们树立一个正确的理念有点困难。10年前每个程序员都曾面临资源有限的状况，所以老程序员总是会想着对程序进行压缩。

编程需要非常大量的精力，所以大多数程序员都比较年轻。这就会带来一个问题，因为编程需要很多的训练。在年轻时，目标不是很持久，可能会被这样那样的事分心。但是年轻的程序员应当坚持下去，他们会变得更出色。作为程序员，我认为自己在1975年到1980年之间的提高是最明显的。在1975年，我会说："嘿，看看，我什么事情都能做。"我真的认为自己可以，因为我读了大量代码，从来没有发现哪段代码是我无法快速读懂的。今天我仍旧认为检验编程能力的最好方法之一就是给程序员一本30来页的代码，看看他阅读和理解的速度有多快。

采访者：你认为那是天赋吗？

盖茨：一定是天赋。有点像是纯I.Q.（智商）。你必须只专注于代码，并把代码和你已经编写的程序关联起来。很多人会说："我要花上好几天来看这些代码。"而一个优秀的程序员则会说："我把代码带回家，晚上花一个小时就能全部看完。"这种能力的差异是巨大的。

采访者：学习计算机科学是成为一名程序员的最佳途径吗？

盖茨：不是，成为程序员的最佳途径是编写程序并研究其他人编写的优秀程序。我自己以前就是去翻计算机科学中心的垃圾桶，找出他们的操作系统的程序清单。

你要愿意去看别人写的代码，然后写自己的代码，再让其他人复查你的代码。你需要身处这个不可思议的反馈循环当中，让世界级的专家告诉你，你做错了什么。你不能让一些小小的个人习性阻碍你获得这些反馈信息。有些世界级的专家会在一些纯属个人偏好的细节上喋喋不休，比如说该怎样注释程序。你必须跳过所有这些东西，因为在某种程度上，他们是试图以自己的形象来塑造程序员，并试图让你按他们的方法行事。而这些可能并没涉及程序的纯质量问题。

如果和一个优秀的程序员聊一聊，你会发现他对他使用的工具非常熟



悉，就像一个画家了解他的画具一样。优秀程序员们开发程序的方式都有很多共同之处，这点令人惊奇——他们得到反馈的方式，以及他们是如何精准地进行规范的，哪些是草率成就的，哪些是认真完成的。当你请这些人来看一些代码时，你会发现他们的反应通常是非常非常一致的。

采访者：是否有人对你编码的方式产生了特别的影响？

盖茨：每个编写PDP操作系统的人都对我产生了影响。还有TRW公司的约翰·诺顿，他在读别人的代码时会写一些备忘录——在此之前我从来没有见过有人这么做。后来我也开始尝试那样做了，看别人的代码时我会写下心得。

我和保罗·艾伦一起编写过很多程序，我们之间总会有些共同的想法。当你调试代码或者不能确定一些特定的权衡时，马上就可以找人交谈，那种感觉真的很好。从某种意义上说，那就像是一种休息，可以缓解紧张情绪，不必转换主题就可以与别人接着讨论。在创作的过程中，能够减轻一些压力，但仍能集中注意力，这样是很好的。我和保罗知道如何能够有效地一起工作。你不会发现太多像我们这样的搭档。他对我的影响巨大。后来在微软期间，查尔斯·西蒙尼和微软其他一些人也影响了我。



采访者：软件行业会出现什么情况？我们会继续做另外一个出色的字处理软件或电子表格软件，还是说计算机行业会扩展到我们今天甚至连做梦都想不到的领域中？

盖茨：我们正越来越多地思考计算机。我创造了“软软件”这一新词，是指随着时间的推移，程序自身会与用户的需求和兴趣相吻合。会出现更好的文字处理器和电子表格系统，我们会使用网络、图像和新的体系结构。并且将使用大容量的存储光盘（CD），可以在上面存储百科全书。

真正不同的将是基于规则的编程方式。其不同之处在于，编程不再是“如果发生这种情况，就这么做；如果发生那种情况，就那么做”。这是程序现在的工作方式。今后你会先写下规则，然后让小的推理引擎查看当前的情况描述和规则。程序将尝试推导出新的情况描述并采取相应措施。例如，程序可能包含了重力规则，如果东西从桌子上掉下来，程序就会知道，如果掉下来的是玻璃的话，可能会摔碎。因此，相对于常规类型的编程，这种编程会以很不明显的方式来产生结果。

所谓的专家系统就是基于这种技术构建的。基于规则的编程是通过证明

机完成推导的过程，而不是在程序中做明确的说明。也许这些技术在今后的三四年或更长的时间内都不会产生影响。但一个想取得成功的年轻程序员会聪明地把注意力放到这种新的编程方式上。

采访者：基于规则的编程的方式在处理差异很大的信息时会比传统的编程方式更有效吗？

盖茨：嗯，这有点儿不太好解释。假设有一个程序是关于如何建造桥梁的，它采集了所有关于金属压力、弯曲度和特性的信息，并在程序中嵌入了有关工程、材料和这类信息的资料。但如果你跑过来对程序说："我们想用塑料来建一座桥。"那么这种变化是巨大的，就好像对它说"我想在火星上建一座桥"一样。

基于规则的编程方式在极端情况下，所有体现了金属可以承担多大压力、重力如何产生作用等物理原则都会作为一条一条的规则明确地阐述。所有的推论都来自对这些规则的检查和运用。目前我们还没有足够好的用来证明规则的引擎，如果采取这种极端的方法的话，效率会低得让人无法忍受。但这正是我们正在取得进展的一项技术，而且这种技术可能很快就会改变我们的编程方式。另外一个想法是，可以让很多台计算机同时并行运行。实际上，这可能会有助于提高这种基于规则编程的效率。这种重大的体系结构变革可能会影响人们的编程方式，或是影响他们对编程的看法。

对程序员来说最可怕的事情是，编译器太好了或是计算机太快了，让程序员变得不再重要了。我过去总是担心，自己选择了某个专攻的领域，但其重要性可能会随着时间的推移而降低。

采访者：微软的涉及面很广，而整个计算机行业的变化又非常迅速。你是如何做到与时俱进的？

盖茨：嗯，我不会想着要跟上每一个变化。我正在和IBM、苹果、DEC和日本计算机界的高层人士合作。我必须知道将要发生什么，我不能浪费太多时间去猜测。我和微软的工作人员飞往某处时，我们会谈论计算机界发生的事情。微软的电子邮件系统是一个高效的工具，可以帮助我跟得上行业的发展。

要跟得上行业的发展，方法之一是使用个人计算机。我要确保阅读了用户手册，并使用了排名前十的软件产品。这些产品不会经常变化，所以我对它们还是很熟悉的。如果你真的关心个人计算机软件，就会使用每一个软件，会去了解它们，并考虑你的软件要如何才能比这些软件包做得更好。



从某种意义上说，个人计算机已经变得简单了。现在我们只有两个体系结构，PC和Mac。而在美好的往昔，我们有三四十种完全不兼容的机器，还有一大堆乱七八糟的语言混在一起。因为我们让很多很多用户使用计算机，所以必须让计算机体系结构变成同类的、更加标准化的，以便用户能够了解其中的一些动向。很多在业内发生的事情并没有推进行业的进展。网络和图像方面的工作可能和业内最先进的东西相关，我们会把注意力放在这上面，而不会去管什么这个零售连锁店会倒闭吗，这家伙有没有行贿那个家伙，这家公司给了那个人足够的股票吗？谁会管这些事情呢？真正聪明的人会把注意力放在自己的领域上，他们会给我提供他们认为有重要意义的东西，他们会带给我能够产生影响的项目。

采访者：微软在10年后会怎么样？

盖茨：我们的目标很简单。我们开发的软件要能够让每一个家庭、每一张办公桌上都放上一台计算机。我不知道这是否要用10年的时间，我不擅长猜测准确的时间表。微软还希望参与到生产更好的计算机的工作中，为计算机开发系统软件，并开发很多运行在这些系统软件上的重要的应用软件。



即使会有越来越多的计算机，我们现在也并不认为需要扩充我们软件开发小组的规模，因为我们可以开发那些销量大的程序。我们可以获得高额软件收入，但公司的规模又不会比现在大很多。这意味着在公司中，我们可以了解每个人，可以一起讨论，分享开发工具以保持高质量的开发水准。

当前微软正专注的一个新领域是CD上的应用软件。CD光盘将是我们用来把个人计算机引入家庭的技术。

采访者：你为什么认为CD的出现会让微机进入家庭，而其他技术却不行呢？

盖茨：现在如果买一台计算机，然后再买一个教育软件，你会发现在使用后并不能达到预期的效果。程序在解答问题的数量上、问题的多元化上以及模拟现实生活中的方式上乏善可陈。随着大容量存储光盘CD的出现，教育软件可以创造一个让你产生直接共鸣的环境，其中包含的大量信息、各种各样的解答、亲身参与其中的感觉，都会给人留下深刻印象。

这是一个竞争的世界。有了教育软件，我们就要与报纸、书籍和电视竞争。而当今市面上的教育软件并没有什么竞争力。除非你只是为了不让孩子变傻，否则还真没有什么好的理由把这样一台机器买回家，因为它不会吸引

你，不会吸引非计算机人员。

采访者：*你觉得在CD光盘上的新软件能够与电视竞争吗？*

盖茨：电视是被动的娱乐。可以肯定地说，人们想要的是互动，是有多种路径可以选择，并且能够从计算机得到他们想学习的内容的反馈。他们可以查找感兴趣的东西。CD光盘这种设备是可以互动的，这种特性使它有别于只能观看的电视。

采访者：*在设计CD光盘上的应用程序时，你们会采用很多与现在相同的设计原则吗？*

盖茨：CD光盘软件是完全不同的。我们希望用户可以借助于CD光盘地图软件查看美国地图，指到某处，点一下，放大地图，然后说："嘿，周围有什么旅馆？"程序就会告诉你答案。如果使用的是百科全书，点到贝多芬的一首交响乐，计算机就会播放出来。这是一个新的接口程序，完全不像字处理器或电子表格那种提高生产力的工具软件。CD光盘上的程序要解决的是完全不同的问题。它和任何一种新媒体一样，有很强的竞争力。如何运用编程技术开发出比别人好的CD光盘应用程序？这是值得深思的。这个市场和我们已开发的程序所面向的市场不一样。在这个市场中，我们希望我们的智慧能有助于开发出一些新颖而又适用的软件。



采访者：*这么说，它并不是简单地把一堆报纸放到CD光盘中，再配上一个检索程序？*

盖茨：嗯，有些人会那样做，但我们不会，那没什么令人兴奋的。我们深信将来在每一辆汽车、每一栋房屋里都会有一个带CD光盘的计算机。当你到了一个新地方，把那张小小的光盘放进去，拖动图像，它就会显示出你要的路线，告诉你感兴趣的信息。

比如说体育运动。放进去一张体育CD光盘，就能看到运动员的记录和照片。可以观看过去的比赛，可以查看比赛规则。每张CD都会有你希望了解的某个领域的信息。每一张CD都会有小测验，比如："嘿，你以为你对棒球了如指掌吗？嗯，这个人是谁，他做了什么？"

每张CD上都有互动游戏，在体育CD上这是很明显的。而在音乐CD上，游戏将是"这首曲子是什么"，你可以查看到得分，查看作曲的人，听不同乐器的声音。你会坐在那里，输入自己的得分。如果你是个飞行员，很可能

会对机场和飞机的图片感兴趣。所有这些我们都会放到光盘上。

采访者：这些CD光盘上的软件会在书店出售吗？

盖茨：最终会的。在起步阶段，我们要确定哪些销售渠道对CD光盘感兴趣。特别是，会有一个专门销售计算机软件的零售商店吗？我个人并不这么认为，但这很难说。

某些CD应用软件听起来像是天方夜谭。但是发明一种新的媒体需要多长时间呢？我们几乎没有发明过新媒体。录像带不是新媒体，它并没有什么特别之处，只是电视节目的延迟播放。交互式视频磁盘本来有机会成为一个新媒体，但它却没有足够的用户群。它没有足够的信息资料，没有低成本的播放器，使用也不方便。它没有融入到大众文化中。光盘是一种超级的互动式视频，但我们必须做得更好。

CD引发的变革将是巨大的。我认为零部件商品目录将不再会以印刷品的形式发行了。对于主要是用于参考的东西，你需要的不只是翻页，还要以一种不同的方式来查找、处理和查看信息，这种电子形式远胜于其他大多数形式。我们最强的竞争对手显然是书籍。我们不会损害图书市场，但CD将取代商品目录和某些类型的参考资料。



采访者：你认为会从CD光盘中诞生出一种文化吗？就像电视文化那样。

盖茨：我不知道这种文化是什么，但CD比电视更具互动性。CD光盘软件不像现在的个人计算机那样只是针对编程的。但是仍旧会像个人计算机一样让人上瘾、让人能够参与其中。你会参与进来，你会想做测验，你会说："我是一个超级巨星，让我试试这个。"我们会提供测验功能，这样可以让多人参与进来，挑选问题让别人回答。如果一个孩子沉迷于个人计算机，我认为这远比沉迷于电视要好，因为至少他需要做出选择。我不是那种讨厌电视的人，但我认为电视不会锻炼人的头脑。我家里现在刚好没有买电视机。

采访者：如果采用不同的标准，你认为CD光盘的发展会被削弱吗？

盖茨：对于微软和那些规模比微软大上百倍的公司来说，建立这一领域的标准需要强有力的政治手段。目前可能会出现两种甚至三种不同的、相互不兼容的光盘阅读器。考虑到编写不同版本的软件的成本，这种做法是不合适的。所以在标准上我们投入了很多的注意力和精力。我们正在努力确保我们的标准就是标准。这是很难的，很有挑战性。我们要非常非常迅速地让所有的活

动都围绕一个标准开展，并且我们必须确保这个标准是合适的。

只有半导体行业才能做出这样的光盘阅读器。那些成本低得惊人的内存和高速处理器，视频和音频芯片，使我们能够生产出多媒体计算机。半导体行业正在发生着奇迹。单说最近两年，用户就在大量地购买各种零部件，因此价格也被降到一个很有吸引力的水平上。

采访者：你觉得CD光盘会与专家系统合并吗？

盖茨：不会，这两者并不相互依赖。因为CD上可以存放相当大的数据库，所以可以把专家系统所需的数据放到CD上。但它们并不是互相需要的。它们有各自面临的困难和挑战。

CD是视频，是音频，是程序，还是互动的，所以要想开发世界上最好的CD软件，需要各种技能组合在一起，这种要求是惊人的。像任何一种新媒体一样，难度很大。当人们第一次看到电视时，觉得比电台好多了，但电视也就那样做出来了。过了很长时间后，才发明了丰富的色彩、所有的动作、三维填料，以及今天在电视上看到的各种特殊效果。就像电视一样，随着我们媒体经验的增加，CD光盘软件也会越来越好。我现在可以坐在这里，告诉你所有我们不会犯的错误；但是5年后，我还可以坐在这里，告诉你所有我们曾经犯过的错误。尽管我们有创意，也无法马上就很充分地利用媒体。



采访者：你可曾希望能再回去动手编程吗？

盖茨：哦，那是当然。编程时可以控制一切。编程时不存在妥协。每一行代码都是你的，每一行你都觉得很好。这么想有点自私，但就像允许做纯数学一样，在编程中能够看到一些东西运转起来。我有时会忌妒我的同事，因为他们只需要关注他们正在编写的程序。

续写传奇人生

比尔·盖茨一直担任微软公司CEO，直到2000年卸任。在盖茨的带领下，微软先后推出DOS、Windows、Office、企业服务器等重量级软件，四面出击，并不断进军新的领域。1984年到1994年，微软凭借MS-DOS和Windows逐渐统治家用桌面电脑操作系统市场，1990年推出的Office最终成为办公软件领域的领导者。1995年到2005年，微软将产品线扩展到计算机网络和万维网，同时进军其他行业和市场：1995年推出MSN门户网站，并将其与Windows

95绑定；1996年与NBC合资创建MSNBC，拓展有线电视新闻业务；2001年推出Xbox进入游戏机市场。在盖茨的领导下，微软市值曾一度达到470亿美元。

2000年盖茨卸任CEO时，转而为自己创造了一个“首席软件架构师”的职位，并继续担任微软主席。2006年6月，他宣布将在两年内淡出微软公司日常事务，并将自己的职责分派给两个人：雷·奥奇掌管日常管理，克莱格·蒙代掌管长期产品策略。2008年6月27日是盖茨在微软全职工作的最后一天。不过，现在他依然担任微软的非执行主席。

盖茨从本世纪初开始，就一直在慈善之路身体力行，他和妻子投入近半家产，创建了比尔·盖茨与美琳达·盖茨基金会。2008年从微软退休时，他又将580亿美元资产捐入该基金会。

在微软以外，盖茨投资的公司有：Cascade私人投资公司、bgC3（智囊公司）、Corbis（数字媒体许可和版权服务公司）和TerraPower（核反应设计公司）等。他还购买了由BBC录制的费曼物理学讲座的视频版权。这些视频现在通过微软的Tuva网站向公众开放。盖茨还曾在纪录片《等待超人》中出镜，该片试图分析美国公众教育的失败之处，并荣获2010年圣丹斯影展“观众选择奖”之最佳纪录片奖。





6

约翰·佩奇



约翰·佩奇（John Page）于1944年9月21日出生于英国伦敦，他从十几岁就开始使用计算机，并在20多年的职业生涯中继续从事着计算机领域的工作。

1970年，佩奇加入了惠普公司。他在伦敦、日内瓦和欧洲其他地区为惠普做了4年多的技术支持工作。他1974年移居到惠普公司总部所在地——加利福尼亚州的库比蒂诺，负责管理HP 3000计算机的全球技术支持。后来，他转入软件研发领域并开发了Image数据库管理系统。在惠普工作期间，佩奇在斯坦福大学学习了人工智能专业，完成了计算机科学方面的研究生学业。

1980年，佩奇离开惠普，和弗雷德·吉本斯（Fred Gibbons）及贾奈尔·贝德克（Janelle Bedke）联手创办了软件出版公司（Software Publishing Corporation，SPC）。佩奇在他的车库里开发了软件出版公司的第一款产品，即后来的PFS:FILE。现在PFS系列中的软件已经超过六种，涵盖了信息管理的各个方面。佩奇是软件出版公司负责研发的副总裁。

佩奇是一个身体修长、健康，稍有点孩子气的人。他目光柔和、笑容亲切，说话带点儿轻微的英国口音。我们见面时，他穿着一件未扣领扣的蓝色衬衫和一条灰色休闲裤。他带我穿过软件出版公司那有着加州风格的舒适的红木梁柱办公室，进入一间很大的空会议室中。在那里，他以一种放松的状

态思考和分析了他在编程和管理软件公司上的做法和心得。

当我要求约翰·佩奇分享他的一段源代码，以便我们把源代码展现到书中时，他拒绝了，他说这个要求在他看来“是一个疯狂的想法。建筑师会发布他们已经建好的教堂和博物馆的样子，但不会发布蓝图碎片”。他认为一个代码示例对于读者来说没有多大意义。他建议我们列出他在计算机领域中的许多成就和贡献。正是这段表述，比其他任何话都更能体现佩奇的编程思想。他在开发软件时总是把普通用户放在最重要的位置考虑。激励佩奇并让他着迷的是最终的结果，而不是实现的手段和途径。

*　　　*　　　*

采访者：是什么原因促使你开发了PFS:FILE程序和PFS的一系列软件？

佩奇：当我被调到惠普在加州的生产HP 3000的工厂时，我参与了大量的软件开发。我在那里开始了Image的工作，那是一个运行在HP 3000上的数据库管理软件，是目前大型机和小型机上应用最广泛的数据库管理系统。



在惠普的时候，我在数据库管理系统上花了很长时间，让我吃惊的是这些系统是为程序员而不是为最终用户定做的。这在当时是合宜的，因为当时各家公司更需要一个开发工具，而不是一款为最终用户提供的产品。但是在观察个人计算机的开发时，我觉得可以开发一个普通人也能使用的数据库管理系统。于是我开始琢磨这样的项目。

那时惠普对个人计算机不感兴趣，就更不用说个人计算机上的软件了。我遇到了贾奈尔·贝德克和弗雷德·吉本斯，后来我们一起创办了公司。我们讨论这个项目，决定找一家软件开发商，却没能找到。后来我们看到软件产业正蓬勃发展，意识到创办软件公司的机会来了，于是就成立了SPC。我们先制定了一个业务发展计划，告诉我们可以预期的增长和市场机会。当我们最终决定成立公司时，开发PFS系列产品的想法在我脑海中已经酝酿了大约一年了。

采访者：你们是如何使PFS从当时市场上其他产品中脱颖而出的？

佩奇：我觉得当时缺少一个普通人也可以使用的应用程序，就像电话或汽车一样。我们的目标是设计一个像家用电器那样简单易学的程序。这是一个有趣的设计权衡，因为这意味着我要追求的是最容易理解的软件，而不是最高的性能和最丰富的功能。这有点像电话系统，从外表看，它不是很复杂。拨

号，电话响了，然后开始交谈。就这样，并不复杂，对不对？但要做到这一点，必须在幕后实现大量复杂的技术。PFS的设计与电话系统相同：外表简单，背后却有着复杂的技术。

在设计PFS时，我偶然发现了一个奇怪的软件设计原则：和想象的恰好相反，复杂的程序远比简单的程序容易编写。复杂的程序容易编写，是因为你把程序的复杂性丢回给用户了，你强迫用户做各种困难的决定。例如，假设用户想知道一个文件中有多少个数据块，你在程序中提供了这个功能，让他自己找出答案。但他用这个信息要做什么，谁知道呢？而如果是一个非常简单的程序，设计者自己必须清楚用户为什么要了解这个信息。从为程序员设计非常复杂的软件到为普通人设计出可用的软件是一个非常有趣的转变。

采访者：为什么要用Pascal来编写PFS呢？

佩奇：别忘了自从VisiCalc被取代之后，PFS是目前市场上使用期限最长的软件了。在5年前，还没有C语言，只能在BASIC、汇编语言和Pascal之间选择。BASIC不行，我也不想用汇编语言开发整个程序，因为花的时间太长。我需要一个功能比较强大的高级语言，Pascal就成了当时唯一的选择。还要记得，那时IBM还没有发明个人计算机，当时市场上唯一值得考虑的计算机是Apple II，而Pascal是那种机器上唯一能够满足我需要的编程语言。



在开发新版本时，我们正在切换到C语言，因为C语言是一个比Pascal更好的开发语言。但对我们来说，Pascal语言并不差，不像那些书呆子程序员所认为的那么差。我觉得C语言的效率只比Pascal提高了5%～10%。

采访者：你说PFS的开发非常困难。那设计和编程一共花了多长时间？

佩奇：我花了大约18个月设计和编写FILE和Report功能。虽然在最后阶段得到了一些帮助，但主要是由我自己开发的。

采访者：开发过程中遇到的最大的困难是什么？

佩奇：当时最大的问题是把整个程序塞进苹果计算机的48K的内存中，并让它以一个合理的速度运行。Pascal程序是解释型的，这意味着程序性能很不理想。程序设计完成后，运行速度非常慢，于是我把所有出现性能瓶颈的部分都隔离出来，用汇编语言重写了一遍。这样问题就解决了，程序的性能总算达到了预定的目标，但是用不带调试工具的汇编语言重新编码，真是一段可怕的经历。

另外一个问题是商业的战略问题。在我们推出PFS的时候，一台48K的苹果计算机被认为是非常高端的配置，应该说是顶级配置，常见的配置是32K的计算机。尽管有64K的苹果计算机，但多出来的那16K计算机内存板的价格是550美元。请记住，当时个人计算机真的是个人自己掏钱买的，人们用自己的钱购买这些计算机，花五六百美元买块内存板可不是件小事。我们必须做出决定。

我们无法让程序运行在32K的计算机上。我们已经把FILE和Report分成两个程序，因为如果把它们合在一起就无法在48K的Apple II计算机上运行。现在它们还是两个程序，尽管用不了多久这种状况就会改变的。一个只能运行在48K计算机上的程序能成功吗？需要更多内存的做法是有风险的，因为它限制了市场的规模。我很多个晚上都睡不着觉，躺在床上想有没有办法突破这样的限制。

采访者：*当时有没有其他数据库程序面市呢？*



佩奇：当时我们非常担心Stoneware公司的DB Master软件会让我们的投资打水漂。不过我们通过细分市场解决了这个问题——DB Master软件被视为是高端产品，用起来非常复杂。在后来的几年里，两家公司的命运出现了分化，这很有意思。

采访者：*当时你预料到PFS会变得如此成功吗？*

佩奇：没有。我在生活中通常的做法是先规划目标，而一旦设定了目标，就不再重新审视，也不会再考虑这些目标有多么困难、是否值得做、是否会成功。我不再考虑别的了，只想着完成任务。我就是这样做完了FILE程序。当它开始卖的时候，我感到非常震惊。我想："上帝啊，它能行，看啊！"

采访者：*你现在还编程吗？*

佩奇：我仍旧觉得编程很好玩，但不能像我喜欢的那样再编写很多程序了。考虑到公司的士气，你不能太深地陷在项目或编程中。在一个大点儿的公司里，领导者需要知道每件事，但又不能多到让员工觉得自己不够好或不再被需要了。公司员工需要觉得他们是在负责自己的工作，并且在心理上觉得他们是正在实施的项目的主人，否则就没有积极性了。

我常常好奇在微软公司工作会是什么样的感觉，因为这个公司的负责人在技术上居主导地位，并且非常出名。我不知道他是否会对公司软件开发人

员的士气造成负面影响。我也常常想，如果有一个人对公司有着完全的控制力，是否会让公司的成长潜力受到限制。当公司发展到一定的规模，创办人不能再面面俱到时，其他人就必须准备担起责任来，按照创办人的思路领导公司向前发展。在SPC，我们正试图远离人们注意的中心，以便公司有更大的发展舞台。

采访者：从一个计算机程序员转变到运营一个庞大的公司，你觉得困难吗？

佩奇：这个角色转换是一个非常平缓的过程，因为我已经做了二十多年的管理人员，我可以告诉你真相，最难的是从管理者重新变成程序员。在创办公司前，我在惠普已经习惯于领导很多人了。再回到编程工作时，我被迫完全依靠自己。真是让人一震。事实上这有点儿可怕——我不知道自己是否还能编程。但是，就像骑自行车一样——你永远都不会忘记的，只要一踏脚蹬，车就奔着夕阳去了。

采访者：现在你自己已不再编程了，那么你是如何招聘到好的程序员的呢？



佩奇：公司需要真正的系统设计师——就是那些能够透过基本的工具看到需要构建什么、需要使用哪些技术以及该如何使用的人。例如，一个聪明的工程师在知道内存将会变得非常大时，他会坐下来反思：这种趋势对于程序设计意味着什么？我因为认为内存非常有限而一直在做哪些假设？如何克服这些旧的想法并将它们彻底消除？如何设计一个反映新视角的方案？一个公司中必须要有这样的人。

公司还需要那些可以为宏伟的设计架构编写代码的人，就是那些好的技术人员。把需要做什么很精确地定义好，就可以交给那些刚从学校毕业的、获得计算机科学学位的人来做。他们都能做好。接下来的问题就是如何在这两类人之间搞好平衡。

采访者：独自一人编程和一个团队合作编程有什么不同？

佩奇：如果一个产品需要四到五个人开发，那和你自己一个人编程时采取的方法是不同的。我坚信只能由一个人来做产品和高层结构的设计，最多不能超过两个人。所有的设计出于一个头脑，才能够实现和谐与一致。如果试图取悦于所有人，让一个委员会来做设计，你的方向就错了。那绝对是致命的。

所以我在定义阶段会用一个人数非常少的团队来工作，然后如果有必

要，再扩大团队来实施编程设计。实施设计方案所需要的队伍越大，就越必须严格地把整个系统划分成可管理的模块，并定义好相互间的接口。除非一个程序大到必须由两三个人以上来做，否则不必编写很多结构化程序报告，也不用为每个步骤都编写文档。

有些系统，例如设计一个控制航天飞机的程序，需要上百名程序员。那就必须严格地把结构划分成可管理的模块才行。一个小系统的架构可能会过于大，一个大系统的架构可能会过于小——使用的技术必须与问题的规模相匹配。我就是这样做的。

采访者：有些程序员没有参加设计，你是如何让他们参与到项目中的？

佩奇：他们可能会感到失望。但在开始阶段要告诉他们，他们必须接受在不介入设计的条件下参与到项目的开发中。幸运的是，你会发现那些新程序员们都想成为一个高水平设计师的好徒弟。这使他们能够进入这个行业，并提供了一个途径让他们能最终实现目标，自己做一些设计。



采访者：你是如何管理一个团队来做软件开发的？

佩奇：当一个团队在我们公司第一次用SPC的方式做项目时，他们需要学习很多东西——我们看待问题的方法使我们的产品与众不同。我从一开始就教他们这些原则。我不会告诉他们应该如何做设计，而是教他们我做设计的流程。换句话说，我试着去教育他们。我试图传授一些他们可以终身受益的技能。有时候，他们会想出一些我不喜欢的东西，有时他们也不喜欢我做事的方式。我们发现一个团队在开发第一个程序时总会有些磕磕绊绊。但我们刻意去经历这一痛苦的过程，因为我希望公司的员工可以支撑公司未来的发展，而不会出现人员断层。这种方式效果非常好。我们已经有一些团队在做他们的第三个产品了，他们做得非常、非常好，做得比我更好。我发现自己从他们那里学习到了很多东西，我从中得到了真正的乐趣。

采访者：SPC考虑问题的方式有什么独到之处吗？有一些程序员必须遵循的技巧或规则吗？

佩奇：你必须了解客户，了解计算机，为客户和计算机设计出最优秀的产品，除此之外，就没有什么基本原则了。

你必须了解客户，知道他们想要什么。这样才能为他们设计出合适的产品。产品可能是一个很简单的程序，比如PFS:FILE，也可能是一个极其复杂

的程序，比如编译器。前者是为普通用户设计的，后者是为程序员或工程师设计的。我不会容忍一个软件工程师只想做复杂的程序设计。首先，我认为这样的态度不够成熟。一个好的建筑师不管是设计一个小凉亭还是设计一个大博物馆，都能从中得到快乐，它们各有各的挑战。第二，我认为真正的挑战是如何设计出一个外表简单而内部复杂的软件。我对那些不喜欢这种挑战的人感到困惑，因为这种挑战对我来说非常有趣。

采访者：你在计算机行业已经工作很多年了，对于这个行业这些年来发生的变化，你肯定会有一些看法吧？

佩奇：计算机还停留在老框框里，因为在相当长的一段时间里都没有出现什么新东西了。市场追捧的是过于复杂的系统，这个问题主要是出在IBM上。所有“新”的功能都来自小型机和大型机。每一代新的IBM个人计算机都比上一代复杂、难用。个人计算机正在迅速向小型机倒退，IBM甚至可能会把它变回一台大型机。个人计算机的优雅正在渐渐消逝，而原来旧系统的复杂性正在复辟。连可怜的老式Macintosh也在遭受打击，因为它缺乏信息管理系统的管理人员所喜欢的东西。Macintosh的设计师可能会被迫把那些复杂的东西加到苹果机上。这种可悲的事情可能迟早会发生。

最终结果是，个人计算机越来越受到那些在大企业里习惯使用大型机的人的欢迎。他们开始接受个人计算机，但仍然试图把它当做一台小型机来使用。但个人计算机不是小型计算机，它应该是一台巨大的个人计算器，我们已经失去了这样的远景。



采访者：你认为这种趋于复杂的趋势会继续下去吗？

佩奇：我希望不会。我有一种感觉，个人计算机将朝两个方向发展。将会有一种商用个人计算机，它会使用与大型机相同的工具，比如COBOL和其他一些可怕的东西。信息管理系统的管理人员仍在使用COBOL编程，就像二十年前一样。如果给他们任何创新的东西，他们会说：“不，让生活简单些吧。我想要我亲爱的COBOL语言。”

我也乐观地认为，计算机行业将回归到制造普通人可以使用的个人计算机，就像原来的苹果计算机那样。它们价格更低、功能更强，并且更易于使用。我们现在并没有进入正轨。发展真正的个人计算机将会给整个行业注入新的活力。

采访者：为了跟上不断变化的消费者的步伐，你们是如何开发产品的？

佩奇：为了适应变化的市场，可以从两个方向调整产品。一种发展是由计算机及其技术驱动的。你会发现一种新的适合于计算机的应用程序，可能和最初的设计完全不同。这种增长会产生一种与最初的需求完全不同的需求。例如，如果增加了排版方面的业务，你会突然意识到个人计算机非常适合做排版。这需要的软件和最初设计的完全不同，并且为了易于销售，软件功能必须强大。

另外一种发展是由客户体验的变化驱动的。在越来越了解到计算机能够做些什么的时候，他们对功能的要求提高了，但并不希望复杂度也随着增加。他们希望新的程序能做更多的事情，但仍能保持简单。我们面临的挑战就是满足这一需要，因为我们选择了这种发展方式。我认为必须通过满足客户的需求来加速发展计算机的技术水平，例如，在不增加复杂性的情况下让程序有更多的功能。这很费脑力，但只有这样，才能有真正的进展。遗憾的是，这个行业痴迷于复杂性，并出于一些荒谬的原因希望把这种复杂性展示出来。



采访者：你认为现在整个行业是由技术驱动而非市场驱动的？

佩奇：是的。他们追求技术主要为了炫耀。这有点像那些昂贵的音响——花的钱越多，音响上的灯和旋钮就越多。如果你跟一个真正的用户，例如买这个音响的人的太太谈一谈，就会发现这真是有问题，这些装置除了把她弄糊涂外，再也没有其他作用了。我敢打赌，如果查查控制器上的指纹，看看哪些按钮真正使用过了，你会发现，九十个按钮中只有三个是使用过的。没有人知道其他按钮是干什么用的，也没人用过它们，因为就算没有那些按钮，音响一样很好用。

人们在计算机程序上也是这样。人们因为要写信或做报表，所以会买一些设计粗糙的字处理器。那些程序中充斥着没有人会用、也不需要的各种控件功能。开发商的诀窍是散布对技术的疯狂痴迷。很不幸，因为SPC刻意向用户隐藏技术细节，经常有人认为我们对技术不够专注。这真的很烦人，我们辛辛苦苦隐藏了复杂性，却因为程序没有显得很复杂而被业界误解。

采访者：你是如何隐藏复杂性的？

佩奇：给你举个例子。在PFS Report中，如果想得到一张好的、让人放心的报表，只需提供在报表中希望显示的四个条目就可以了：公司名称、地址、

销售额和销售员的名字。程序通过搜索数据库找到列的最大宽度。然后通过一个启发式例程计算出每列的空间和边沿的宽度比例。程序中的启发式例程非常复杂，开发的过程是很费脑筋的。这个例程会做出各种尝试，直到为每列找到“满意”的比例，并做出排列和选择。这样报表看起来美观而均衡，两边的空白幅度相同，宽度足够，列与列之间的间隔也不太远。

程序采用了纵向制表方案，确保报表在页面上是居中垂直的。这个比例既考虑了美观，又能完整显示数据。我们来回挪动数据，尝试不同的布局，直到数据在报表页面上让人看着合适为止。这些报告在99%的时间看起来都非常棒，而那剩下的1%也是完全可以使用的。最终结果是，大多数人只需指定报表中的条目而不必去考虑布局，就可以得到一个美观的报表了。

而不够先进的报表程序要求用户指定每列的宽度、每列在纸张上的位置。他们强迫用户去思考“姓名”这一列要多宽。用户就猜呗——他想不到有人的姓名会超过一定的宽度，比如说，25个字符。结果可想而知，有一个姓名被切掉了。用户耗费大量的精力确定数据应当放在报表页面的什么位置，其结果很糟糕，因为用户无法直观地看到报表是否匀称。



但那却成为很抢眼的程序。难用的程序反而成为卖点。复杂性变成了一个功能：复杂系统的用户可以精确地设定每列的位置，而PFS却把这些列放到了合适的位置。实际情况是，几乎所有使用复杂程序的用户都要更改五次才能让报表看起来恰到好处，否则就只能接受一个布局很糟糕的报表。我怀疑这怎么会有效率呢。程序设计问题变成了控件功能与生产率之间的问题。要么是高生产率，要么是大量的控件功能。这样的取舍是程序员和用户一直面临的问题。

采访者：听起来开发一个简单易用的程序似乎需要更多的时间。你为什么能够投入这么多呢？

佩奇：这样做有利于用户，也让我们能够区别于竞争对手。构建一个自动传输程序肯定比构建一个手动传输程序要难，但那是客户需要的。现在在市场上的程序有两百万个，我们相信这些额外的投入对我们也是有好处的。

采访者：你如何能不断地想出编程的新思路呢？

佩奇：有时连我自己都觉得惊讶，但真正的答案是与客户保持联系。我需要原始的数据，我要自己来解释这些数据。因此，定期地，在一天的某个安静的时候，我会通过我们的业主登记卡随机给用户打电话，询问他们为什么会

买我们的产品，他们正在用这些产品做什么事情。人们对这种人性化的关注非常震惊，但他们通常会告诉你很多信息，告诉你他们是怎样使用产品的。

我们密切关注竞争对手，但不会被行业的集体思维所诱惑。我们会参加一些同业人员不会参加的集会和会议。我们喜欢那些不是完全以计算机行业为主题、但又很可能会讨论到计算机的会议。投资方面的会议的参加者往往涉及多种学科——半导体产业、硬件、软件和销售的从业人员。我们通过开展“采纳经销商建议”的营销计划来访问经销商。汇入每个人的观点就形成了该行业的多元化。

运气也起到了重要的作用。软件出版公司的名称是非常容易误导人的，因为我们并不出版软件。但是，几乎任何一个想要销售程序的人都会把程序送到我们这里来。于是我们看到了很多来自全国各地的新想法。

最多的构思来源于公司内部。我们使用自己生产的软件，我们要求员工尽可能使用自己的产品，以便从他们的使用过程中得到反馈，不论是成功的地方也好，失败的地方也好。由于有200名员工使用程序，我们得到了很多反馈数据。当然，实验室里的工作人员也提出了他们自己的想法。



采访者：你和其他程序员的互动多吗?

佩奇：我们尽量和软件人员一起度过一些时间，因为他们之中有人能够想出好主意。但出人意料的是，我们之间的互动并不多。我们有一些非常聪明的人。但是很奇怪，大多数程序员往往知道自己的本事，却不知道如何使用这些本事。他们倾向于聚在一起讨论程序清单之类的事情，一直讨论到凌晨。我的反应是觉得这样很无聊。我会想：“嗯，不错，有点儿意思，但却不是生活的本意。”

我没有遇到太多能够和我一样关注如何解决问题的人。我觉得郁闷，我遇到的都是建筑工人，却从来没有遇到过建筑设计师。我希望有办法能够结识计算机以外其他学科的设计师——航空航天、建筑施工或是工程。我知道其中会有很多相似之处，虽然我不知道那些相似之处是什么。将来的某一天，把这些人都聚在一起开会，会很有趣的。

采访者：你认为计算机在未来将扮演什么样的角色?

佩奇：计算机在所有地方都将扮演绝对核心的角色。信息是万事万物的基本构件。事物的信息几乎和事情本身一样有价值，而计算机处理的正是信息。我正在读一篇发表在《商业周刊》上的很有意思的文章，就是关于信息的威

力。人们开始认识到好的信息是多么重要。有一个投资额达数十亿美元的人说，他真诚地相信，如果用实际的钱数来衡量，那些关于他的投资的信息比他的投资本身要有价值得多。

想想吧——这意味着信息是像黄金一样贵重的资源或原料。仔细想想如何使用它、处理它、保护它。这正是计算机能够出色完成的。计算机将会渗透到社会的各个层面，这个预言肯定会实现的。

采访者：计算机将在哪些特定的领域中发展？

佩奇：我觉得有两个重要领域还需要10到15年才能发展成熟。一是让个人计算机回到正轨上，让它们真正成为个人使用的——大小像一本书，价格在300美元左右。这将在两年内实现。

第二个领域是通信。目前，信息移动的方式很原始——又慢又繁琐，而且费用昂贵。因为竞争和新技术的原因，更好的方式正在开发中。从这个角度看，AT&T的解体是令人回味的。



采访者：计算机对通信领域的影响会是什么？

佩奇：通信会更快、更便宜。信息只有在能够得到时才是有用的，而你现在却得不到。人们没有意识到他们的信息多么贫乏。一旦他们认识到存在很多的可能性，他们都不会相信自己曾经生活在没有完整信息的状态下。

信息的存取将对我们的生活产生深远的影响。例如，以后可以通过数字化语音进行通信。还记得你第一次给我打电话时的留言吗？你的电话打到了我的电话应答机上，那就是一台计算机。你的声音被数字化后，存储在服务于整个办公大楼的计算机的磁盘上。我可以把留言转给别人，也可以变更这些信息，并从家里转发过来。只要使用按键电话就能拨号进入系统。没有计算机，这些都是做不到的。我们很快就会推出一个家用版本，在300美元左右——应答机之间能真正地沟通。那才是吓人的地方。

采访者：你认为我们未来会访问什么类型的信息？

佩奇：我举个例子，说明可以提供什么信息。假如你想去欧洲某地度假。你想要有优质的饮用水，一个带游泳池的漂亮的海边酒店，而且在附近可以乘滑翔机飞行。但究竟怎样才能找到符合所有这一切的地方呢？你可能会在旅游小册子上找到一个好的海边酒店，但他们可能不会提供你最感兴趣的那些东西的信息，像滑翔机和游泳池。除非碰巧看到一张照片，否则你几乎得不

到任何资料来确定度假地点。因为找不到新的、更好的度假地，你最终只能回到你每年都去的那个地方度假。还有很多其他例子，几乎在生活的所有方面我们都缺少大量的信息。

采访者：现在是否已经有一些这样的信息了？

佩奇：已经有一些了。举例来说，我是个飞行员，因为可以登录气象局服务系统，把气象简报下载到计算机中，所以我的飞行能力提高了20%～30%。而另外一种获取天气预报的方法，效率就低多了。打电话到气象服务站，电话可能会被搁置半个小时。如果真能接通，也可能得到不正确的信息，或者我自己可能会记录错误。而我只要在计算机中输入请求，气象信息就出现了，又快又准确。有意思的是，我获得的信息比航空调度员的信息还要好。我直接查询科罗拉多州博尔德国家气象局的气象信息，而航空调度员得到的气象信息是由那些联邦政府没钱更新的、过时的系统提供的，所以我得到的信息确实是要好一些。现在我仍然需要打电话预订飞行计划，但这很快也会改变，很快就可以在线预订飞行计划了。



采访者：为什么会有那么多程序员都把开飞机当成是一种消遣呢？

佩奇：一个原因是那些相当成功的程序员是有财力开飞机的。还有一个原因是，飞行对工程师来说是一个锻练，因为它涉及操纵复杂的事物，这是工程师喜欢做的。程序员也喜欢驯服复杂的东西，而开飞机是非常复杂的，所以掌控飞机是很有趣的。最后，开飞机像编程。一名优秀的飞行员要给乘客提供完美、平稳的航行体验，而优秀的程序员给客户提供的是在计算机上完美、无需费脑的经历。

采访者：你还有其他兴趣爱好吗？

佩奇：我正在盖一座房子，我弹吉他，喜欢游泳，不是为了游泳而游泳，而是为了锻炼身体。

采访者：听起来你并不像是一个工作狂啊！

佩奇：我不是工作狂，只有当我去做那些非常具体的工作，并且像我刚才所说的那样，不再考虑其他事情时，我才会是个工作狂——我已经有一段时间没有那样做了。身处领导者的角色，我发现不能长时间地工作了。领导是一种人际交往活动，需要在正常工作时间内完成。同时也是非常消耗精力的，付出很多却没有回报。当身边的人在成长、新的项目开发成功并走向市场、

公司持续增长时，你会看到间接的效果。这些令人心情愉快，但却没有编程的那种成就感所带来的直接喜悦。由此，我觉得工作时间不能太长。我必须为自己留出些时间来，这样才能保持平衡的状态而不至于发狂。

采访者：作为你所说的那种不再考虑其他事情的程序员，是否有可能会失去平衡性？

佩奇：你应该要告诉所有的亲友："瞧，我要离开6～9个月，虽然我人还在这儿，但心却不在了。我得去干那个工作，所以会心不在焉。希望你们能理解和容忍，我保证等过了这段日子、完成工作以后会补偿你们的。"如果是对你亲爱的家人，履行这一承诺是很重要的。这样拼命地工作，可能会对你的婚姻和其他的关系造成极大的破坏。

有些人觉得完成项目真是太过瘾了，他们做完一个项目就接着又做另一个了。这样会把自己累垮的。如果连续开始新的项目，情况会变得很糟。

另外，当你在复杂的程序上艰苦工作时，锻炼身体是很重要的。大部分程序员都缺乏身体锻炼，这样会失去敏锐的思维。往往在连续完成了第二个或第三个工作后，身体就虚弱了，以至于你会产生幻灭感，你会对着镜子说："上帝啊，看看我，我为什么要这样做呢？"



采访者：为什么程序员们都会沉迷于工作呢？

佩奇：你在脑海中不断地考虑正在做的整个系统的状态。如果失去了脑海中所思考的形象，就需要花很长时间才能重新回到原来的状态。这就像是一名空中交通管制员，在他脑海中有9架飞机，他清楚地知道每一架飞机要去哪里。如果这时候打扰他一下，问他什么时候下班，他就会失去那些飞机——那些在他脑海中飞机的控制模型。在编程处于最佳状态时，有一个大的复杂模型是非常有效的。一旦从那中间出来了，就得工作相当长一段时间才能重新进入状态。

采访者：你编程有惯常的做法吗？

佩奇：就个人而言，我在早晨的工作状态是最好的。我喜欢很早就起床，那时很安静，适合写程序。我尽可能在上午做那些需要思想高度集中的工作。尽量把会议安排在下午，那时我的精力不如早上了，但我仍能好好地谈话。到了晚上就很累了，无法创造性地解决问题。所以从晚上6时到第二天早上，我会毫无使用价值。

采访者：你编程的过程是什么样的？

佩奇：我会坐下来想想程序要做成什么样子，然后在脑海中描绘出程序的部件。我倾向于首先聚焦在那些我认为可能会有问题的地方，并想办法将它们弄清楚。这地方看起来很难，那地方看起来也很难，而剩下的不过是些普通文件和很熟悉的散列表。在对最难的部分单独处理之后——也许是编写一个小程序来证明一些理论——我对整个程序就有了一定的信心了。有些部件很简单，有些很难，但我知道该如何处理。然后在开始实现之前，我会定下整个程序的结构。

我必须相信所要做的是可以实现的，否则就会心烦意乱。我见过一些不成熟的程序员，他们很害怕达到最终的目标，他们只关注了程序的某些部分，马上就开始编码了。他们是从一个相对次要的位置开始编程的。

在画好结构草图后，我就一个部分接一个部分地去做了，并确定各部分之间的接口。我不喜欢那种让人不得安宁的感觉，就是我设计了东西，却不知道那些重要的部件是否可以构建出来。这种感觉让我胆怯，让我无法充满信心和活力十足地开发项目。



采访者：如果程序的某些部分不能工作，你会把整个程序废弃掉并重新开始吗？

佩奇：在付出很大努力并明白了为什么在我之前没人能做出这个东西时，我会觉得兴奋。如果能解决这个问题，我就能做出一些大家都认为不可能做到的事情。这种想法让我热血沸腾、心跳加快。我喜欢这种挑战，就好像一只狗得到一块骨头——我是不会把它放下来的。我会一直思考这个问题，不管是开车、游泳还是沐浴。我就是要跟这个问题死磕，最终找到一种解决办法，也许会使用一些没人能想到的技术。当然，如果最终证明这个问题是完全无法控制的，我也会放弃。但通常，经过漫长而艰辛的思考，是会找到问题的解决方法的。它可能不是一个优雅的解决方案，也可能不符合所有的计算机科学规则，但如果方案可行，管那么多呢！

我一直在说，计算机科学不是一门真正的科学，因为你没有实际发现有关物质世界的任何东西——但是，这只说对了一半。有时候一个问题的答案一下子豁然开朗了，就好像答案一直都在那里一样，而我发现了它。我仔细检查，发现答案是完美的。怪吧？

以PFS:FILE的设计为例。我记得自己坐下来，做了四五个差异很大的设

计。有些用的是关系方法，有些用的是自然语言的方法。我提醒自己一直在寻找的标准是什么。我一直在寻找一种方式，能够把设计方案扩展到一系列的程序——报表编写器、字处理理器和图形方案当中。产品系列之间要非常相似，它们各自的工作方式也不能让人感到与众不同。与此同时，程序的功能必须齐全。程序必须要既简单又容易地做我让它做的每件事。我合成思路并对设计尝试了不同的方法。有些设计在一些领域工作得相当好，但在其他领域却不行。每种设计都是不同程度地满足了所有不同的标准。然后，突然地，就好像是我偶然碰到了一个设计一样，它能够满足所有的目标。

有时，在开发了一个很好的程序后，我会觉得是我发现了它。做完了之后，我看着它，觉得只能那样做。这几乎是无法控制的——解决方案就这样让我撞上了，没有什么别的办法。这些感觉表明我对那件事情已经了解了。是的，发现的过程就是这样的，但这种情况并不常有。

续写传奇人生



佩奇当时是软件出版公司（SPC）负责研发的副总裁，他主导开发了PFS系列产品。1986年，SPC发布的Harvard Presentation Graphics产品大获成功，公司也将重点转向高端商务软件。PFS系列产品最终在1991年卖给了Spinnaker 软件公司，而佩奇也早在1990年1月就离开了SPC，此后媒体上关于他的信息就很少了。

至于SPC，产品上的转型却最终导致了公司的失败。到1993年，虽然Harvard Presentation Graphics占SPC总收入达80%，但它在Windows市场上却远不如 Lotus Freelance 和 Microsoft PowerPoint。1996年，SPC被Allegro New Media收购。

美国计算机历史博物馆网站上有佩奇在2006年讲述的关于SPC创立和第一个产品诞生的回忆。2008年，《IEEE 计算机历史年鉴》（vol. 30 no. 2）上刊登了佩奇撰写的关于SPC创业的故事。

```
PAGE 1  TRIMMERS.C 10/23/85 9:9

/*=================================================================*/
/*

strdcpy - copy string until a specified character is found (or entire
       string is copied)

    returns a pointer to 'chr' in from string (or terminator)
*/

char    *strdcpy(from, to, chr)
char    *from;
char    *to;
char    chr;
{
    char    c;

    while (c = *to = *from) {
        if (c == chr) {
            *to = '\0';
            return (from);
            }
        to++;
        from++;
        }
    return (from);
    }
/*=================================================================*/
/*

strtrim - trim off blank characters (and CR, LF) from right side of line

*/

strtrim(line)
char *line;
{
    int     i;
    char    *p;

    i = strlen(line);
    p = line + i - 1;                    /* RHE of line */
    while (i--) {
        if (*p == ' ' || *p == '\t' || *p == '\n' || *p == '\r')
            *p = '\0';
        else
            break;
        p--;
        }
    return;
    }
```

莱特莱夫用C写的两个子程序，里面用的就是他在dBASE III里的编程风格。



7

C. 韦恩·莱特莱夫



从1969年到1982年，C.韦恩·莱特莱夫（C. Wayne·Ratliff）在Martin Marietta公司工作，担任了工程和管理方面的一系列职位。当“海盗”号空间飞行器于1976年在火星着陆时，他正是NASA（美国国家航空航天局）“海盗”号飞行团队的一员，为“海盗”号的着陆支持软件编写了数据管理系统MFILE。

他在1978年开始编写名为Vulcan（火山）的程序，并在1979年到1980年期间自己做市场推广。在1980年下半年，他和Ashton-Tate达成了市场推广协议，并把Vulcan产品重命名为dBASE II。在1983年年中，Ashton-Tate购买了dBASE II的技术和版权，莱特莱夫也加入Ashton-Tate成为新技术的副总裁。莱特莱夫是dBASE III的项目经理，同时兼任设计师和主程序员。

莱特莱夫生于俄亥俄州的特伦顿，成长过程中辗转于俄亥俄和德国的多个城镇。他现在住在洛杉矶地区。

我前往格伦代尔的Ashton-Tate研究中心与dBASE的创造者C.韦恩·莱特莱夫交谈，他在宽大的办公室里欢迎了我。我们坐在一张圆桌旁，就他的成就以及他对于编程的深刻见解交谈了很久。莱特莱夫是个高大的美国西部人，有一种特立独行又让人感到舒心的风度。在计算机界待了十五年多之后，他身上仍然洋溢着冲劲和激情。和很多已经厌倦实际写代码的程序员不同，他仍然每天精力旺盛地参与开发的所有阶段。

*　　*　　*

采访者：什么使你成为了程序员？

莱特莱夫：上大学时，我曾设计一辆小的双人座、后轮驱动的汽车。那是在六十年代，车子很大很快。于是，我开始使用一台CDC 6400计算机来辅助设计，因为我想从真正工程的角度来看待汽车的设计，而不是凭空猜想可以给我的设计加上多大的发动机。我写了一堆小程序来帮助设计悬挂机构，找出重心，诸如此类。没过多久，我就开始写其他类型的程序，因为我从编程中得到的快乐已经超过我在制造汽车中得到的快乐。

采访者：所以编程让你脱离了汽车设计领域？

莱特莱夫：是计算机本身把我从汽车设计领域拽跑了。在我完成学位之前，我从丹佛市的Martin Marietta公司[①]得到了一份工作，当一名计算员。我的头衔是计算员。其他人是用计算机编程，而我直接就是一名计算员。[②]

采访者：你不妨解释一下。



莱特莱夫：嗯，那要回到航空和航天工程的早期年代了。当人们需要，比如，解一个微分方程时，他们会让一大群人用门罗计算器[③]，每人负责一个独立部分。一个人做一组加法，然后把纸传给下一个人，做乘法，就这样一直下去。那些人就被称为计算员。计算员那时很像行政助理，不过做的是工程方面而非行政方面的工作。因为我会编程序，他们就让我干这个活了。然后，在1969年的越南战争的热潮中，我应征入伍。

采访者：那你去越南了吗？

莱特莱夫：没有。由于在Martin Marietta工作，我有了军队文职人员所需的技能，于是我继续干编程的活。有两年的时间，我的工作是开发一个名叫LOGEX的后勤战争游戏，使用COBOL开发。我的大部分工作和设备与补给的订购相关。那就像是给一家非常庞大的公司做库存，只不过有一些军事方面的特殊规则需要处理——比如，使用核武器需要由总统批准。

采访者：有什么特别的事情让你去写Vulcan程序吗？

莱特莱夫：退役后，我成了Martin Marietta公司的雇员，承包（航空航天局）

---

① 一家美国公司，在1995年与Lockheed公司合并，组成著名的洛克希德·马丁公司（Lockheed Martin）。

② 计算员和计算机在英文里都是computer。

③ 门罗（Monroe）是一个手摇和电动计算器的主要品牌。

喷气推进实验室的工作。我是“海盗号”项目的成员，写了“海盗号”着陆器的数据管理程序，名叫MFILE。在1976年吧，我开始对自然语言的设计和实验感兴趣，所以我买了IMSAI 8080的8位机套件，并且自己组装。那足足花了一年，主要时间花在等待各种配件上。我焊了2200多个焊点。当然，如果可以用同样或相近的价格买到整机，我会去买的。

等到把所有东西装到一起，我就有了一台计算机。除了1K的内存，其他什么都没包含在内。你必须继续购买其他东西，比如键盘。我已经在套件上花了1000美元，然后我要花159美元买键盘。最终我大概花了6000美元。现在你可以买一台完整可用带硬盘的AT计算机，也只要6000美元。

采访者：你对于自然语言的实验是不是就成了dBASE的基础？

莱特莱夫：可能你会觉得奇怪，dBASE其实始于橄榄球博彩。我靠选择赢家和比分差值进行投注。我对橄榄球一直了解得不多。是赢钱的愿望，而不是橄榄球比赛，吸引了我。我那时认为，如果花大力气进行数学处理，我可能会赢。



具体的方法是研究所有的统计数据。每周一早晨，报纸上会刊登有周末博彩的所有统计数据——这至少要占一个双版。在赛季进入第四、第五周时，我有一整间房间堆满了报纸！我试图从这些报纸里挑选出赢家，而这真是一个可怕的过程。我最后确定，没有计算机的话，这些数据实在是太多了。不过，世事变化快，不到一个星期，我就完全忘掉了橄榄球。我得出结论，世界需要一个自然语言的数据库管理器。很明显，必须用到计算机，而这就是我最初购买IMSAI 8080的原因。

我到外面买了很多自然语言和人工智能的书籍。我从一个领域走向另一个领域，期间也做了很多实验。此前，我对数据库和自然语言处理都了解不多。

我决定在数据库管理器的基础上进行自然语言的工作。自然语言本身是一个非常具有不确定性的课题。你需要有一个场景才能够让自然语言发挥威力。我开始考虑我为“海盗号”项目写的数据库管理程序，打算在8位机IMSAI上重新实现一遍。很巧，我看到了一个叫JPLDIS（全称是“喷气推进实验室显示和信息系统”）的程序的描述。它很容易理解，非常简单，非常干净。我立即想到，在微型机上实现它会很容易。

拿现在的说法，我相信JPLDIS其实是IBM一款叫做Retrieve的产品的“克隆”。Retrieve在某些分时系统上运行，所以，对我的程序Vulcan来讲，从

Retrieve到JPLDIS已经是个进步了。我打算先对付数据库部分，然后用自然语言的方法进行改进。我推迟了自然语言的部分，到今天都还没有开发出来。

采访者：*那dBASE程序的早期版本，或者叫做Vulcan，是什么样子的呢？*

莱特莱夫：我采用JPLDIS的概念，裁剪了其规格要求，写出了Vulcan。JPLDIS可以处理200个域，但我觉得16个就已经足够了。这样，我让它运转了起来。在开始这个项目一年多以后，我就用它来处理我的个人税务了。我觉得Vulcan还是有些商业潜力的，于是就开始对它进一步改进，以达到可销售的地步。在1979年10月，我开始进行市场推广，并在《字节》[1]杂志上登出了第一个Vulcan的广告。在四五个月后，我又登了一个四分之一版广告。广告的反响远远超出了我能应对的程度。

采访者：*所以反响在一开始就是积极的。那时候谁是你的竞争对手？*

莱特莱夫：FMS 80，后来还有Condor和Selector。在我写Vulcan代码的1年零9个月里，我的软驱坏了两次。每一次都要花3个月维修，才能让机器重新跑起来，所以我浪费了6个月。我一直在想，如果能提前6个月，我可能就成了第一个。



采访者：*所以，你突然间就有了一个成功打入市场的产品。这个成功让你惊讶吗？*

莱特莱夫：我完全超负荷了。我得自己干所有的事情。当订单来时，我得把订单打出来，填写发票、打包程序、制作一份磁盘的副本——所有的一切。我自己投放所有的公告，我还得不断加工程序。我结束工作回家，还得继续工作直到半夜，精疲力竭地睡去，第二天起来，然后重复这一过程。那时我需要对Vulcan做大量的改进工作。几个月后，我彻底累坏了。

1980年夏天，我决定不再给Vulcan做广告，就让它的影响逐步减小直至消失。我会继续为前期买家提供支持，但我不打算再主动出击寻找新的买家了。

采访者：*你为什么不试着把它卖给一家大公司，或者雇些人手帮忙？*

莱特莱夫：我没想到。那时候并没有什么大公司。就我所知，每个人都可以独当一面。华盛顿大学的一位教授和他夫人考虑拿下销售这一块，这时，乔

---

① *Byte*，七八十年代非常著名的计算机杂志，1998年停刊。

治·泰特（George Tate）和霍尔·拉什利（Hal Lashlee）打电话过来了。他们到我这边来，看了演示。即使演示中出了些问题，他们仍然表示理解——他们知道什么是演示，对此我印象深刻。大部分人观看演示时，一出现错误他们就没兴趣了。乔治和霍尔已经开了一家公司，叫做Discount软件公司。他们有一名雇员——他们说起来似乎他们有一堆雇员一样，但他们实际上只雇了一个人。另外，他们住得很近，离我只有10到15英里远。看起来像是天作之合。他们开了个价，要求独家销售权，我就接受了。我们就以这种方式合作了两三年。

采访者：写Vulcan时，你想到它会这样成功吗？

莱特莱夫：我在不同的阶段有着不同的想法。在开始自己销售之前，我幻想着如果《字节》杂志的读者中能有十分之一购买Vulcan的话，我就可以退休不工作了。这个想法没持续多久。订单并没有如潮水般涌来，于是我调整了自己的期望。即使在我和Ashton-Tate达成协议之后，我在日记里的记录也只是我期望在这笔生意里获得100 000美元。总共。我想这说明了我的志向已经变得有多保守了。



采访者：让我们现在讨论一下从Vulcan到dBASE的转变。你在Vulcan里做了哪些改进后产生了现在的产品？

莱特莱夫：有些改进是在用户界面上，还有一些是在性能方面，不过主要改变是把打字机般的显示方式改成了全屏的方式。在我看到DamStar之后，我意识到每个人都想要以二维的方式使用屏幕。

我添加的新命令几乎完全是在用户界面上。我觉得还没来得及做所有需要的改进。按我的理解，今天人们对dBASE的感觉是程序不够用户友好。回过头想一下，如果跟着感觉走，而不是听从别人的意见的话，dBASE今天会更加接近完美。

采访者：但它已经是一款非常成功、颇受好评的产品了。你是不是有点完美主义了？

莱特莱夫：我收到过各种各样的建议：让它更大，让它更快，让它更小，切到16位，支持多用户，支持多语言，法语、英语、荷兰语、德语……方向太多了，我每个地方只能满足一小点。最终来说，我觉得这是我的过错。

我的愿望是让程序更强大、对用户更友好。如果我坚持这么做，程序今

天会更好，但我不确定它是否会更成功。也许我得到的建议是正确的。在很多方面，我很难想象有什么方法能使程序在销售量上更加成功。

采访者：你认为dBASE为什么这么成功？

莱特莱夫：很大程度上是靠运气。不过，dBASE也确实生逢其时，在正确的时间出现在了正确的地点；那不完全是运气，而是靠设计。它既是一种语言又是一个数据库管理器，这一点现在看来极其重要。

dBASE满足了用户的想象，因为它是非常开放的。我的编程人生就是制造工具。即使在10年前，和其他程序员比较时，我发现自己很大程度上喜欢通用化地解决问题，而其他人则更习惯于写程序解决某一特殊需求。他们的程序常常比我的交付得更快，但我的程序则生命期更长。当特殊需求改变时，他们的程序也就完蛋了，每次都需要重新写。我写程序的方式总是让程序能解决一类问题，而不是单独一个。



dBASE和BASIC、C、FORTRAN、COBOL之类的程序也不一样，因为dBASE已经做了很多脏活。数据操作是由dBASE而不是用户完成的，所以用户可以专注于他要做的事情，而不是跟肮脏的细节较劲，比如打开、读取、关闭文件，管理空间分配等。

采访者：所以，从一开始，你设计dBASE时就想着用户的方便性？

莱特莱夫：哦，对极了。人们经常问我这样的问题："我应该这么做，还是应该那么做？"我本能地就会考虑："用户需要什么？他们会怎么用？这给他们带来什么好处？"非常多的程序员只考虑该怎么写出程序，他们从来没有做过销售。他们可能是好的程序员，但他们没有想过真正的市场需求。

采访者：从这方面讲，你是不是建议程序员不要那么技术驱动，而要多考虑用户和市场？

莱特莱夫：是的。你认为黑客这个词是好是坏？如果你认为这是一个好的词，那就可以把他们叫做黑客。他们是好人，但他们只想着他们自己的工作，而不是用户。你需要走得更远一点。dBASE就是这样开始的。我考虑的是我希望怎么使用这一产品。

采访者：你看到了自打你进入这一产业后的巨大变迁。你的编程技术有没有随着时代发生一些改变？

莱特莱夫：改变不是源自我学到的东西，而是源自计算机的改变。最开始在

计算机上干活时，我总是要制作穿孔卡，然后成批运行。我会拿着卡片跑到操作员那里，请求他提交我的作业，然后回去，每半小时检查一下窗口，看看程序是否已经运行、打印出了东西。一般而言，我会在纸上写出完整的程序，不断编辑和删改，然后交给打孔操作员，把卡片打出来。那可是一锤定音：你得写出整个程序，确保它们在纸面上就是正确的。在计算机上运行就是让它们工作。没有渐进修改一说。

今天，使用CRT显示器，你可以自己键入程序。有了个人计算机，你几乎可以立即作修改，但个人计算机不如大型机强大，编译和执行要花较长时间。所以，我倾向于一次写上几行，试一下，让代码跑起来，然后再写几行。要完成大的实质性改变时，我总是力图在每个迭代里做尽量少的工作。所以，现在编程更多是演进式的，而非那种一锤定音式的。

采访者：*最终产品是不是通常会和你的最初设想有很大出入？*

莱特莱夫：在某些方面是的。我总是在开发过程中获得新的想法和建议。当我现在规划一个项目时，它最终总会做到我最初设想的一切，另外再加上很多其他的东西。要不就是，当我开始把玩这个程序时，我看到了改进的机会。我总是不放过任何一个改进的机会。



采访者：*你现在还写程序吗？*

莱特莱夫：没有我希望的那么多。我现在需要处理很多业务方面的事情。

采访者：*是的，看来很多写了成功程序的程序员最终都离开了他们的最初根基，转而进入了管理层……*

莱特莱夫：对我来讲很大程度上就是这样，但我打算逐步减少甚至完全消除参与管理的时间。我希望这样，因为管理消耗了极大量的时间。我在办公室时写不了什么程序。我不停地说话，可能是电话，可能是面谈，可能是开会。我所有的程序差不多都是在家里完成的。

采访者：*在家庭生活和深夜的编程工作之间你是怎么平衡的？*

莱特莱夫：我通常在我的妻子卡罗琳允许的时候才去做编程项目。在我结婚前，更准确地讲是在两次婚姻之间，我会工作到午夜。我喜欢在晚上工作，因为晚上没有干扰。电话不会响，邮递员和园丁也不会来。四周很安静，因而有时间可以集中思想。

采访者：你有没有确定一种你认为真正有效的工作方式？

莱特莱夫：不论是单枪匹马还是在一个非常小的团队里，我都能工作得很好。当团队超过6个人时，那就完全失控了。泰德·格拉瑟（Ted Glasser）有很多专利，他是计算机界的元老级人物，曾入选《名人录》，他有一次告诉人们他可以管理的最大团队能装在一辆大众车里开出去取比萨饼。后来他改了口——现在变成一辆普通的美国轿车了。我完全认同这个观点。

遇到多个项目，我一般只做一个，完全排斥其他事情。如果我切换的话，那就是完全切换。我可能把一些事情搁一边，这些事情就会靠边站，可能我几个月都不会再去碰，然后我才会回过来重新处理。这是我唯一可以进行多处理的方式。开始做一个项目时，我真的希望做到底为止。我可能没法做到彻底完成的状态，但我会至少做到一个可接受的状态。要我把进行中的事情靠边，一定要有好的理由才行。要有更大的原因，比如罪恶感。罪恶感非常有效。如果我觉得我违反了对别人的承诺，我真的会把自己更乐意做的事情放到一边。



我是那种喜欢做点计划的程序员，但我并不会把计划做到事无巨细。我有关于目标的一个想法，但真正的工作是找出能达到目标的下一步。我总是做能达到下一步的最少的工作。在一个步骤里，我取最小的子集。我不是先做最难的部分，也不是先做最容易的部分。这并不是一个数学上严格的步骤，而是由情感和直觉来决定的。

采访者：那你是不是认为自己属于注重细节的那类人？

莱特莱夫：不是，我处理细节只是因为我知道它们必须处理。这不是我乐于做的事情。我的意思是，有些事情，比如让程序提高几纳秒的性能，我觉得完全无关紧要。如果程序运行速度只达到了预想的一半，你知道再过一年，有了新的机器，它就会运行得非常正常了。

采访者：编程有什么地方让你感到满足？

莱特莱夫：唔，改装跑车对我来说是一个令人满足的项目，因为我可以做些什么，让它出现在我的面前。这个项目最糟糕的问题是，我需要的某些零件根本不存在。而在计算机上，如果我半夜里需要什么东西，这个东西目前我还没有，我总可以自己做出来，不管它是什么。最糟糕的情况也不过是需要很长的时间才能做出来。

当然，这不是我喜欢编程的全部原因。我喜欢高科技的东西，我喜欢做出一些东西，让它们显示在屏幕上。如果你程序写得好，它可以非常优雅：它就像在歌唱，它制作精良。从工程的角度来看，我享受面对这一切，就像面对一辆漂亮的汽车，一座坚固的桥，或是一栋精美的建筑。它的一切都是那么平衡、那么和谐。

采访者：你能不能再详细说说这种平衡和优雅的感觉？

莱特莱夫：平衡有很多种形式。代码应当干脆精炼。你应当可以用一句话解释任何一个模块。如果可能，内容应当按字母顺序排列。仅从视觉角度讲，代码不应当在任何地方掉出打印纸的边缘。不应当有一个很大的“if”，而“else”却很小。一切东西在任何地方都应当平衡。平衡是这里的关键字。

采访者：当你写代码时，代码第一次出来就是平衡的吗？还是需要做很多修改？

莱特莱夫：我会做很多修改。我喜欢拿写代码与黏土雕塑相类比。你拿起一块黏土，然后刮掉一点，然后加点黏土，然后再刮掉点。隔一段时间，你可能会觉得一条腿看起来不对头，所以你就把它拿掉，再装一条新的上去。期间会有很多次的交互。

理想的模块应该有一页长。如果它超过一页，我就必须重新思考：我现在做的是什么？我现在做的不同事情有哪几件？它们该不该进一步分成不同的模块？优雅和平衡的部分要素就是，在一个程序的蛋糕一样的层次结构里，要在某种程度上让所有的模块都具有相同的重量、相同的大小、相同的责任和相同的功能。



采访者：平衡对程序有什么帮助吗？

莱特莱夫：平衡使程序变得可维护。当你发现一个好的平衡点时，就好像你发现了什么基本的底层物理原理，并且依据它去执行一样。当东西失去平衡时，你就知道有什么事情做错了。很可能是某种内在的错误使其失去了平衡。通常来说，当某一模块过大而我有很不对劲的感觉时，我会考虑一下我所做的事情，并且重新安排，或者说，重新调整一下各个部件。

采访者：有什么特别人物对你的编程产生过影响？

莱特莱夫：加里·梅耶（Gary Meyer）有一本书叫做*Software Reliability*（软件可靠性），我印象很深。其他书籍也在一些简单的方面对我影响很大。把

东西按字母序排列是我从编程风格的书里学到的技巧。这让事情变得简单，使得程序变得干净，是走向优雅的一步。就是要绝对地简单：我自己都无法想象我为什么自己没有想到这一点。

另外一个影响我的人是我在Martin Marietta公司时的一个老板，菲尔·卡内（Phil Carney）。那时候我写FORTRAN，需要行号时，我就按顺序选下一个。你思考问题的顺序和程序的顺序会不一样，所以我就随便选择行号。看见我这样做，他大发雷霆，说："把这些东西排好，从一百开始，然后每次加十。"我觉得这么做非常有意义。那是很小的事情，但是小事情很有帮助。

采访者：你对编程感到过厌倦吗？

莱特莱夫：当我发现自己在一遍又一遍地重复着同样的事情时，我会感到厌倦。我记得在JPL一遍遍地写同样的循环时，我差不多也有过这样的感觉。每一次做的时候，每件事都会有那么一点点不同，但循环基本上是一样的。然后有一天，我在翻看*Structured Programming*（结构化编程）一书，恰好看到了这个结构化编程设计的流程图，我就想，那就是我需要的。确实如此。



采访者：你会在程序里写很多注释吗？

莱特莱夫：事实上，不怎么写。在公司里，我因为写的注释不多被人批评过。我发现注释可以分为两类：一类是解释显而易见的东西，比无用还糟糕；一类是你需要解释非常复杂、有很多关联性的代码。嗯，我总是力图避免写出复杂的代码。我写程序总是力争把代码写得稳固、清晰、干净，哪怕是需要额外多写五行。我一般认为，你需要写的注释越多，你的程序就越糟糕，肯定是哪里出了什么问题了。好的程序并不需要很多的注释。程序本身就应该是注释。

模块应当小一些。当一个模块超过一页纸时，就是哪里出问题了。每一模块的顶部的确需要一行注释，用一句话解释一下这个模块是做什么的。如果你无法用一句话解释，一定是哪里出问题了。

采访者：杰出程序员具有什么样的特质？

莱特莱夫：编程的范围很广。有些程序员的工作完全是面向用户的，而有些程序员则是解决数学问题，对用户可以说毫不关心。说来奇怪，写游戏的人大部分和用户比较一致。对那种程序员来说，游戏的数学问题可能会让他烦躁不安，而天平的另一头，写一个更好的求平方根算法的程序员可能就会对

决定把东西放在屏幕的顶部还是底部感到无趣。我想把我自己放在靠近游戏程序员四分之三的位置上，更靠近用户界面这一边。

你不可能在整个范围里比较，但你可以比较同一位置里的高下。西奥多・斯特金（Theodore Sturgeon）[①]说过，每样东西都有90%是垃圾。这离事实相去不远。这可能稍有点悲观了——可能该是60%、70%或80%。有清晰的证据表明，在任何可分辨的人群子集里，3%的人做了10%的工作；而另一头，50%的人只做了工作总量的30%。这对橄榄球四分卫、程序员、新闻记者和任何人群都适用。总有少数人可以做普通人能做的工作量的6倍之多。所以有好的程序员也有糟糕的程序员，就像各行各业都有好的员工和糟糕的员工一样。

采访者：程序开发过程让你感到痛苦还是愉悦？

莱特莱夫：两者都有一点。编程有点像参军。身在部队时，我对其深恶痛绝。每一分钟都感到害怕。我一直期待着能够出来。但是当我出来后，拥有这种经历还是很美妙的。写程序时，我享受一路上解决小问题的快感；而当我做完后，虽然经历这一切让人感到愉快，我还是希望在每单位时间里都完成更多。所以，一直会有一点小小的痛苦挣扎。

编程中我最享受的时刻是当我差不多要完成某件事时。我尝试第一次，结果总是很糟糕，后面也会一直失败，直到大概一百次时才会变得相当好。那时会有一种巅峰体验，因为我知道自己成功了。我只需再稍稍加点油，把剩余的bug清除掉。

采访者：程序员的工作是不是很个人化，别人看着你的代码就会说“这是韦恩・莱特莱夫写的”？

莱特莱夫：嗯，SuperCalc的作者加里・巴雷森（Gary Balleisen）曾经用过dBASE早期阶段的源代码，他声称他能从代码风格中看出是谁写的。我知道我的某些做法，比如结束一个代码块的方式，和这里所有人都不一样。我确信我的办法是目前为止最好的。

比如，我第一次看到C时，就不喜欢大括号突出在代码的左边。这让人感到糊涂。但是很偶然地，我在Digital Research的某个文档中看到一种技巧，说可以把这些终结标记跟缩进的代码放在同一层次。唔，这是另外一个可以

① 美国科幻小说作家（1918—1985）。



产生很大影响的小改动。我一直在劝说别人这才是正确的使用方式。这点上我并不太成功。我甚至修改了一个程序，一个美化C代码的程序，来自动做。我可以拿别人的代码过来，在这个程序里运行一下，缩进的风格就成了我喜欢的样子了。

采访者：你有没有探索过人工智能领域？

莱特莱夫：一开始我真的专注于人工智能了。那是一年多以前，我转向人工智能，因为我认为那是未来的方向。不过，我现在已经不这么想了。

人工智能是有未来的，但不会那么快。首先，有自然语言的问题。如果你有一个自然语言系统，你买回家装在计算机上之后，需要用几个星期甚至几个月的时间来教会它你的特定词汇的意义。同一个词在不同的语境下会有不同的含义，甚至一个看起来很简单的词，比如profit（利益），也会有不同的含义。你需要根据你的行业、你的文献等来明确定义词的意义。这一冗长的机器训练过程毁掉了人工智能，使其不能成为开箱即用的产品。



不过另一方面，非常有意思的是专家系统。我的预测是，在两三年之内，专家系统不必再和人工智能牵扯在一起。这就是人工智能的历史：当某件东西开始变得众所周知时，它就独立出来了。模式识别曾经被当作是人工智能，但现在这是一个独立的领域。这也正是专家系统当下的命运。我认为专家系统在我们产业里将变得非常重要，就像垂直应用一样。

采访者：在程序方面，你是否认为将会发生一些大的改变，影响我们使用和处理程序的方式？也许新的语言会变得非常简单，对用户非常友好。让每个人都会自己编程？

莱特莱夫：编程方面会有一些演进。晦涩难懂的语言特性会逐渐消失。像dBASE这样的语言有些方面也很糟糕，但随着时间的推移，这些问题会在演进过程中逐步消失。我们总可以双手合十，祈祷突破的发生，就像电子表格最初出现那样。我希望将来会有同样优雅的东西出现，但这完全是不可预测的。

近来我完全被一个叫yacc的UNIX程序迷住了。它可以帮你创建一个语法解析器。你用巴科斯-诺尔范式（这是计算机科学界广为接受的形式）描述你的语言，把规格说明放到yacc里，它就会生成C语言的模块，可以用来解析符合规格说明的语言。我非常希望对高级程序也能这么做。理想情况下，你可以以规格书的形式把程序写出来，然后编译一下，结果就是你要的程序。这样，你很可能可以在一个月的时间里写出dBASE这种规模的软件。

采访者：你想过退出编程工作吗？

莱特莱夫：没有。在可见的将来，我对编程充满了期待。我妻子偶尔会梦想，我们摆脱所有的计算机，和我们的马一起共度时光。我就会说："等一下，那可不是我要的。"我想要摆脱压力，但我不想要摆脱编程。

采访者：完成dBASE这样紧张的项目之后会不会有结束后的空虚感？

莱特莱夫：那就是你需要考虑新项目的时候了。程序总是在一开始充满着乐趣，你会很快想出无数的主意可以让它做些什么。一开始是零星的小点子，然后你会不断在上面加上其他的功能。当兴奋劲儿过去，你不得不开始写代码时，事情就变得困难了。

我过去认为这个行当里每个人都是设计师。我会跑到乔治·泰特的家里喝啤酒和吃比萨饼。很多时间里，霍尔、乔治和我会在开始喝啤酒前开上半个小时的会，在这半个小时里，我可以制定一整年的工作计划。每个人都想做设计工作。至于实现，那个需要一整年的工作，还有其他杂七杂八的事还是让别人做吧。

采访者：但是你完全不是这样做的。你又写代码又设计。

莱特莱夫：是的，但很多设计是个持续进行的过程，而不是灵光一闪，实现就自然而然地出现。现在让这个公司纠结的问题之一，就是找出最好的方式来做软件。

不管出于什么原因，他们觉得不可以相信个体，或者是一小群人。现在的流程是：市场部搞清程序该做什么，然后告诉开发部他们认为该怎么做，然后开发部花上几个月写一份非常详细的规格书，描述他们认为他们听到了什么，然后公司里的很多人开始评审这份规格书，协商到底应该做些什么。然后困难的工作就结束了。从那时候开始，剩下的只是编码。

这种流程在你造桥时也许是合适的，因为你很精确地知道桥是干什么的。桥总是从河流的此岸伸展到彼岸，而你可以事先精确规定它的最大载重量等之类细节。事实上，对于桥，我可以想象你能在一张纸上说明所有的规格。这也是我认为一个计算机程序所需规格的数量：一张纸的内容。

公司里大家一致认为当前的流程并不好，或者说，太令人痛苦、并不值得去做。最好是找到一帮有点子的人，从门缝底下给他们塞钱，让他们不受妨碍地工作很长时间，在他们认为他们已经完成时，你可以让其他人来摆弄这个程序，并提出改进建议。



另一件很重要的事是让用户试用一下程序，确认它解决了用户的实际需要。这正是我们公司做错的地方，它在忽视用户。公司有兴趣的只是营销，而他们真正感兴趣的交战对手就是其他营销人员。所有的广告都是营销人员做的，并没有用户说这是我们要的。这只是一个营销人员运用他的能力和其他营销人员竞争，所争战的内容只是纸上的规格而已。我觉得这是错的。你在广告里可能看起来不错，可那又怎样？dBASE能卖得好，靠的并不是广告——虽说事后想起来，最初的一些广告做得真是不赖。

采访者：最初的广告活动是什么样的？

莱特莱夫：那是霍尔·珀卢克（Hal Pawluk）做的拿dBASE和水泵相比的广告。在刚写出来时，真是一则相当具有煽动性的广告。

采访者：为什么？

莱特莱夫：广告的第一行写道："我们都知道水泵吸水[①]。"广告里提到的那种水泵的生产公司给乔治·泰特写了一封信，说他们不喜欢把他们的水泵放在贬义的环境里。乔治就说："好的，那我们就在广告下面放一行小注释，标明图里的这种水泵不讨厌。"出于某种原因，他们同样不喜欢这个广告。



采访者：那个广告确实不错。我猜，那时候做生意没有像现在那么死板。你是不是会怀念那类东西？

莱特莱夫：哦，当然。事实上，我现在和公司相处得并不是很好。我们有很多问题。我是个创业者，也就是说，我喜欢从一无所有开始做出某种东西。有些人喜欢一开始有个小东西，然后在成长期把它扩大；有些人喜欢拿很大的东西开始，把它培育得更大，也就是所谓的成熟期。有三个阶段。我想任何两个相邻的阶段还是可以相互协调的，但我感觉我是处于第一阶段，却非得和第三阶段协调。这样造成了很多的摩擦。

采访者：这个分析不错。这似乎也是业界常常发生的事情。公司经过迅速成长，到了一个较大的阶段，人们不得不学会如何应对，会感到沮丧……

莱特莱夫：但软件并没有改变，只是公司改变了。有证据说明公司的态度是错误的。当然，需要有商人来运作公司，但从某种程度上，晚进入这个产业的人并没有领会什么才是软件。他们是在生意行业里，而不是在软件行业里。

---

① 英文中的"吸水"（suck）也有"讨厌"的含义。

采访者：在你这些话的上下文里，什么是软件？

莱特莱夫：理想情况下，那是一种让计算机做对人们有益的事的方法。我并不是说我毫无私心，利他主义也不是我的目标。我的目标是写出优雅的软件和有挑战性的程序。解决社会需求不是我的使命，但确实是软件的一个美妙的结果。

采访者：你认为计算机编程是艺术、科学、技能、手艺，还是什么其他东西？

莱特莱夫：我认为编程有科学的成分，也有艺术的成分。计算机越人性化，里面的艺术成分就越多。游戏开发人员主要处理的是屏幕上的艺术形式，他们是在用高科技的产品来表达那种艺术。这种情况就跟你要做雕塑也得懂得黏土的化学组成一样。不过，你只需要学习几分钟黏土的知识就可以开始进行雕塑，而要胜任计算机上的工作则需要很长时间。不仅是时间，还需要热情。

采访者：dBASE后你会做什么？

莱特莱夫：机会的范围总是在变。在三个传统的生产力工具里，也就是数据库、字处理和电子表格，已经没什么空间容得下新的进入者了，虽说总会有些新的竞争者出现。我们可以看一下Paradox，他们试图成为数据库的领头羊，或至少要得到一些市场份额；还有Javelin，它想干掉1-2-3[①]。现在我不确定谁会被干掉。没有哪个字处理器现在具有优势市场份额，但有好几家的市场份额已经很大了。在Microsoft Word、Multimate、WordStar、Word Perfect、Samna和其他五百家厂商的夹缝里，已经没什么新的字处理器的成功机会了。

仍然有很多的机会，但不在那些领域里。我想，人们在专家系统方面会有很多的成功机会。不能仅仅是系统框架，还得有知识库，可以处理很多类型的问题。有了这样的专家系统，你就可以进入到许多垂直市场里去。

采访者：你认为可能设计出一个可用于多个垂直市场的专家系统吗？

莱特莱夫：可能。至少你可以一次对付一个。即使有一千个垂直市场，每一个也有几千个潜在买家。即使你一次对付一个，理论上说，如果你写出了一个好产品，在一个小的垂直市场里你可能可以拿下几乎所有买家，因为他们



① 指Lotus 1-2-3，那个年代电子数据表的王者。

的行为会很具有从众性。

采访者：你认为IBM会继续占据统治地位吗？

莱特莱夫：是的。在我看来，Apple已经全面坐失良机。在企业产品上他们已经失去了机会，我认为这最终也会影响他们的家庭产品。我认为大家对于Macintosh的潜力已经开始感到幻灭了。AT&T似乎什么也没干好，其他公司也不像有很大的企图。让IBM处于最高位的，我想，是那些兼容机。

实际上，我喜欢目前的市场态势：竞争使得IBM不断进步，而竞争者知道他们如果不向IBM挑战的话，就会受到伤害。如果这种状态能够持久的话，就再好不过了。在和IBM兼容的产品里，虽然有些产品在很多方面更为优秀，厂商仍需要清楚地看到IBM是他们的标杆。我乐于看到他们从此全都过上幸福美满的生活。当IBM在大型机行业过分鹤立鸡群的时候，我真的很讨厌IBM计算机。



那些360、370、3030x和4340系列，我全都不喜欢。我很高兴看到Microsoft和IBM签订了协议。我担心IBM会自己开发操作系统，而我已经见识过了IBM开发的操作系统。

采访者：你有没有在回顾过去时对所发生的事情感到惊奇？

莱特莱夫：哦，是的。有很多事情原本可能会大不一样。我清楚地记得每一次新产品发布前我会恐惧和不安。在Vulcan时期，有一天我工作到很晚，有人给我打电话说："你有没有听说过DataStar？"我说我没有听说过。"有一整版的广告，"他说，"是MicroPro的产品。"我那时只卖出了三四十份Vulcan吧，我想，无论如何有销量就好，但现在看来我要完蛋了，因为我的资源不可能和他们的相比。于是我感到非常沮丧。但过了段时间后，它就消失了。然后下一个是InfoStar。之后MicroPro修正了DataStar里的问题，又是一次大的广告活动，大的这，大的那，然后我想，噢，好吧。同样的事情在Knowledge Man上又发生了一次。当Knowledge Man发布时，我已经接近完成一些自己设定的目标，大概百分之六七十吧。我那时想，新东西要出来了，我要完蛋了。

这样经历很多次之后，我开始不那么担心了。我对R:base并不怎么担忧，对于ANZA我就更不烦恼了。担心并没有什么用，该来的总要来。

采访者：你见过Paradox吗？

莱特莱夫：没有，我没看过任何一款竞争产品。在上周的一个聚会上，我和

市场部的埃里克·金（Eric Kim）和戴夫·赫尔（Dave Hull）交谈，他们问我对Paradox有什么看法，我说我还没有见过它。“什么，你还没有见过？”他们问道。我说：“我没有见过这些产品中的任何一个——我甚至没有见过R:base 4000。”“你没有见过R:base 4000？”他们说。我们回顾了历史和所有那些我没见过的产品。他们就是无法相信我没有见过它们，没有把玩过它们，没有从它们当中得到过启发。

采访者：现在你在做什么？

莱特莱夫：近来我在修改一个公有领域的套件，把它从Pascal翻写成C。这是一个设计和文档语言，差不多是PDL的翻版或者超集。PDL的全称是Program Design Language（程序设计语言），是加州帕萨迪那的Caine，Farber & Gordon公司提出的。它非常优雅，因为感觉像在写程序，但结果并不能在计算机上实际执行。这是一种做规格说明和设计的方式。

采访者：你对今天的年轻程序员有什么建议？



莱特莱夫：如果有人想要编程，编程就不是难事。如果他们不想编程，那无论他们怎么努力尝试，最好的结果也是很难，更大的可能是他们会感到幻灭。所以我的建议是，做你想做的事。

我见过有人一开始并不从事编程或者计算机的工作，他们进入这个行业后就喜欢上了。他们真正被吸引住了，并且真的很快就真正理解了计算机的本质。我也见过其他人尝试进入这个行业，但他们就是做不到。

采访者：这里有什么魔力？你认为是什么让有些人理解了计算机的本质？

莱特莱夫：嗯，这里存在一个组合。我过去常常玩一种心理游戏，问我自己，如果我早生一百年会做什么？我不知道，但一种可能是做侦探，因为编程时有很多侦探性的工作，特别是在调试时。你需要依赖提示和线索。编程的一个好的方面是，你涉及的是清楚的现实，而在实际侦探工作里，你很多时候都无法得到清楚的答案。在编程里，所需的一切只是辛勤工作。只要足够努力，你总可以找到问题的答案。我把自己看成是多面手，我所做的事情，我都认为很重要。有比我更好的程序员，有比我更好的调试员，有比我更好的设计师，以及其他很多方面比我更好的人。我不能当着文雅人重复一个军事教官有一次对我说的话，不过，话的意思就是：“我是万金油，啥都行，啥都不精。”

续写传奇人生

时任dBASE III 项目经理、设计师和首席程序员，亦是Ashton-Tate的新技术副总裁。

1984年8月11日，Ashton-Tate的创始人乔治·泰特因心脏病发作逝世，10月CEO戴维·科尔辞职，新上任的CEO艾德·艾丝柏与莱特莱夫的关系颇为紧张。几个月后，莱特莱夫退出Ashton-Tate，加入了Migent公司。

1988年，莱特莱夫开发了Emerald Bay，首创了客户端/服务器数据库管理概念。这个项目曾一度前途未卜，Migent公司也被迫于1989年关门，随后莱特莱夫自己重新获得了该产品的控制权，并继续开发新版本。

20世纪90年代中期莱特莱夫退休。2007年的一篇访谈中透露，退休后他的兴趣转向数学、航海和与船相关的事情，包括机械、焊接和编程。

莱特莱夫创造的dBASE是第一个广泛用于微型计算机的数据库管理系统，它被移植到多种平台，包括DOS、Unix和VMS。1984年推出的dBase III和后来的dBase III Plus在市场上获得了空前的成功，Ashton-Tate也成为当时世界上仅次于微软和莲花的第三大软件公司。然而自微软的FoxPro之后，dBase渐渐走向了没落。



REFERENCE CARD 3 January 1979

← → - Moves the cursor.

space - Changes the direction indicator.

> - Absolute move. Requests coordinates of where you want to move; you end coordinates with ↓.

! - Recalculates all values (to force an extra recalculation).

label entry - Start with a letter (A-Z), or ", and end with ←, →, or ↓. Use ESC to erase last thing typed.

value entry - Start with a number (0-9), +, or -; end with ↓ or, if appropriate, ← or →. Entry references may be used wherever numbers are allowed; type coordinates or point with cursor. Evaluation is left to right with no precedence. ESC erases last thing typed.

/N - Sets entry to N/A value.

/B - Blanks out entry; erases what was there.

/OO - Sets order of recalculation to C (down columns) or R (across rows).

/OR - Sets autorecalculation to be auto (A) or manual (M)

/OC - Sets size of columns on the screen (must be >0 characters). End with ↓. If screen is split, only affects part of screen with cursor.

/R - Replicates entry. Needs range (such as A2-D2) specifying where to place copies (followed by ↓). If replicating an expression, it will ask for each entry reference whether it should not be modified (N), or should always refer to entry in same relative position (R).

/S - Screen control, sets split screen (H for horizontal, or V for vertical) at cursor position, undoes split and makes it one screen (1), sets label areas at cursor position (L), or undoes label areas and returns screen to normal (N). L prompts for Horizontal (H) or Vertical (V) or normal (N)

; - If screen is split, moves cursor from one half to another.

布兰克林设计的VisiCalc早期版本（当时叫做Calculedger或CL）的参考卡片，这是他在租来的IBM电动打字机上写成的，用在给苹果公司做的一次演示中。本书附录还包含VisiCalc早期的其他若干纪念物。



8

丹·布兰克林



费城人丹·布兰克林（Dan Bricklin）出生于1951年7月16日。1973年，他从麻省理工学院（MIT）毕业并取得电气工程和计算机科学学士学位，毕业后先后供职于数字设备公司（DEC）和Fas Fax公司，从事编程工作，随后进入哈佛大学商学院。在哈佛期间，他集中同学和教授的专长和建议，设计了一款电子表格程序。

1978年，还在哈佛大学时，他跟MIT的老同学鲍勃·弗兰克斯顿合作，开发了这个程序的实用版本。这就是后来的VisiCalc。他们创办了一家名叫Software Arts的公司，该公司于1979年1月注册成立。同年4月，他们跟Personal Software签约，由该公司负责VisiCalc的市场运作。（Personal Software后来更名为VisiCorp。）关于VisiCalc的新闻很快铺天盖地。截至1981年5月，VisiCalc的销量超过10万套。1983年，累计销量突破50万套。Software Arts的成功一直持续到1984年，随后，为了争夺VisiCalc的相关权利，他们与VisiCorp陷入旷日持久的官司当中。

1985年5月，布兰克林离开Software Arts，加入莲花公司，做了很短一段时间的顾问。之后他创办了Software Garden，这是他新开的公司，1985年11月正式注册成立。公司推出的第一款产品叫“丹·布兰克林的演示程序”。

我前往波士顿郊区他的家里跟他会面。他专门腾出一间卧室，用作自己

新开的一人公司Software Garden的办公室。怀揣Software Arts起起落落中悟到的新理念，丹·布兰克林决定从头再来。这一次，他不再浑身充满创办第一家公司时的天真、无限热情和惊人的精力，而是带有一定程度的谨慎、远见和克制。他坦言自己并不想经营农场或看管牧场，而是想耕作一片软件花园，就像后院里的花园，一片足以满足他的需要，并能从中获得快乐和满足的花园。

显然，此前创办VisiCalc的经历对他影响至深。他说话温和可亲，脾气随和，聪明机智。在我们的谈话当中，他一直以审慎、反思的态度，探讨着开发VisiCalc电子表格程序和运营Software Arts积累的经验，并分享了他对新程序和新公司有哪些目标和期望。

*　　*　　*

采访者： 你在MIT学习时是怎么迷上计算机的？



布兰克林： 1970年初，我进入MIT学习计算机方面的知识。我在Project MAC也即现在的计算机科学实验室谋得了一份工作。在那里遇到了鲍勃·弗兰克斯顿、戴维·里德（David Reed）和其他程序员，他们做了很多好东西。

我整个本科阶段一直在那里从事编程工作。我的第一个项目是个计算器。令人惊讶的是，MIT分时系统Multics并未提供命令行计算器，比如输入“Calc 2 plus 2”（求2与2之和），计算器会返回“4”，或者sine of *X*（求*X*的正弦）之类的，于是我自己写了一个。我们楼上的人工智能小组正在开发LISP。当时MIT各色人士云集，比如很有名的黑客理查德·斯托曼。此外，我在MIT还接触到一些来自许多不同领域的资深专家。

采访者： 你在MIT还做其他什么项目？

布兰克林： 我做的项目五花八门。我是APL实现小组的成员，到了1973年，我负责管理实现APL的项目。另外我还参与过LISP项目。

1973年秋天，我在DEC的电脑排版组Typeset-10得到一份工作。我本来已经在他们的语言组谋得一个职位，我受过语言方面的训练。不过，在DEC的面试中，我恰好碰到在MIT共事过的同窗迈克尔·斯皮尔（Michael Spier），他建议我考虑一下排版组。我父亲是个印刷工，我祖父早先也是个印刷工，

这么一来我觉得排版组比语言组更有意思。排版与实际应用关系更紧密。我们有显示屏和电脑排字机。

采访者：你在Typeset-10组主要做什么？

布兰克林：我的第一项任务是编写通讯社转换程序。收到的电讯稿会被转换成计算机排版系统可以理解的语言。藉此我对现实世界的系统有了深入了解。如果程序存在缺陷、延误印刷，导致报纸无法按时上市，报社就会浪费大量钱财，这是我的亲历亲闻。所以我们力争如期完成任务，压力很大。

报社的人一心只想着做好本职工作。他们对技术不感兴趣。只要技术行之有效，他们就心满意足了，他们并不在乎技术本身。我也借此了解到非技术人员对技术的看法。

我记得一家报社在电台上打广告，说他们会提供水门事件录音带的文稿，而用电报传过来的水门事件录音带采用的是这个程序无法处理的某种格式。为了及时把文稿赶制出来，我们必须马上修改程序。同时，这也是我们第一次通过电报读取录音带。我记忆犹新。这件事是在我正准备飞到另一家报社的路上发生的，因为这个问题，我被叫了回去。这真是非常刺激。我回来的时候口袋里只有一把牙刷，因为当时所有行李都已运往加拿大。

这家报社可是个现实世界，压力巨大，相比之下，编译器开发小组或在MIT那会儿则要宽松许多，那里人们对问题都很宽容，碰到问题总是说："好吧，下个月我们会把它修复好。"

另外令我印象深刻的就是认识到真实用户的处事方式不同于程序员。一旦有什么东西变慢了，报社的人就会肆意把这样东西调度到更高优先级，而这会打乱计算机的整个调度算法。他们很奇怪为什么东西会变得"稀奇古怪"。

举例来说，他们的系统控制台用的是一台33型电传打字机，有一次，一小片纸卡在了读取打印副本的光学字符读取器中。这么一来，因为这片纸，排版程序会在每一页上发现更多错误，并提示"格式错误"。但是，这些错误信息都会输出到操作台上，而操作台又跟不上，因为它是台廉价、低速的33型电传打字机。整个系统就这台机器拖了后腿。

总之，尽管他们用的这套电脑系统价值百万美元，但整份报纸就被这张小纸片耽误了，因为他们用的33型电传打字机速度太慢。于是过了一两天，他们就去弄回来一个每秒能传输30个字符的终端，一切都恢复正常了。



采访者：Typeset-10项目结束后又做了什么？

布兰克林：接着我参与了DEC第一款字处理产品的开发。此前我在DEC写过排版终端程序，在另一个排版终端上做了部分软件工作之后，我开始从事字处理开发。我要编写非常底层的微码，实际上给一台内存有512字节的机器写了启动微码，这些微码要烧录到ROM里。随后我投身于PDP-8和字处理开发。

采访者：这款字处理器具备什么功能？

布兰克林：我们在硬件上有很多限制。基本上，我们必须使用标准的DEC硬件。我们可以对其中一个DEC终端稍加修改，让它支持上下滚动，但也仅此而已。我们无法像其他大部分公司那样，使用特制硬件进行字处理。我担任项目主管，带领一个小组设计字处理系统。它是台PDP-8，内存有16k字，每个字12比特。就功能而言，我们的产品跟现在的WordStar也有得一拼。



我们有出色的邮件合并功能，还有表处理和后台打印功能。我们可以编辑软盘容量大小的文档，因为它并不是常驻内存的编辑器。我们全员参与设计，我拟定字处理器的实际规格，同时还编写文件系统、命令系统和后台打印机的代码。

这次经历让我在如何将产品推向市场方面获益良多。设计电脑排字机让我认识到有效利用屏幕和尽量减少击键次数的重要性，因为许多排字工都是按击键次数收费的。这些认识对我后来设计字处理器起到了重要作用，对我开发VisiCalc也很有帮助。

采访者：这期间在DEC的工作跟开发Typeset-10时一样压力很大吗？

布兰克林：在DEC，我通常从早上11点一直工作到晚上1点。我穿着破旧的蓝色牛仔裤，上班时胡子拉碴、披头散发。

DEC搬到新罕布什尔州，我不想跟着搬过去，于是开始另谋出路，跟猎头接洽。我强烈意识到自己应该去拿个MBA学位，这样我在职场上就会更抢手。另外，我也察觉到，当程序员没有前途，他们要跟我这样的毛头小子竞争；新人受过良好的训练，肯接受低薪，愿意工作更长时间。我看到要保持巅峰状态异常困难。我发现程序员到了50多岁再找工作困难重重。另外，我一直想自己开公司，觉得商学院会给自己提供适当的训练。

我申请了哈佛大学和MIT的商学院，两所学校都录取了我。起初我打算

去MIT，因为课程时间较短，我想自己应该快进快出，不过最后还是选择了哈佛，因为我觉得有太多东西要学，最好还是花上两年时间。趁着还没到商学院的空档，我又打了份短工，这份工作很有意思。

我到一家规模很小的公司负责编程工作，这家公司制造基于微型机的电子收银机。这些收银机使用摩托罗拉6800处理器。他们在其中一块主板上放了64KB内存。它用各种同轴电缆把所有收银机相互连接在一起。在这机器上编程用的是一种FORTH变体。我在现有系统上做了大量工作，维护并添加若干新功能，升级系统以兼容新的硬件。

收银机的用户群主要是在快餐店工作的青少年。如果机器出了问题，技术员到场可能需要一整天，店主可不想快餐店因此关门。这个系统非常复杂。每天晚上，总店通过电脑轮询所有店铺，并以电子方式转储这些信息。这样他们就知道货架上还剩多少黄瓜片。客户点巨无霸时，它还知道配方。

我亲眼见证了一家真正的小公司，也可以跟NCR[1]之类的公司竞争并存活下来。这跟在DEC的经历完全不同，那时DEC的销售额刚突破10亿美元，并跻身财富500强。而我已经为上商学院做好了准备。



采访者：你进商学院之后才想到VisiCalc的构思吗？有什么特别的事情激发了你？

布兰克林：在商学院，每当需要编写BASIC小程序辅助自己做功课时，我就会使用DEC系统。但它不够快。尽管我只要花15或20分钟快速写个小程序，就能完成小组项目所需的分析，那还是不够快，而且程序经常出错。

就在那时我想到了VisiCalc的构思，它可以把字处理的即时性和全屏处理的灵活性结合在一起。

采访者：你最初是怎么设想VisiCalc的设计的？

布兰克林：最初的设计充满未来感。我会把手放在计算器上，计算器底部有一个鼠标状的轨迹球，你可以四处移动这个轨迹球，将光标定位在屏幕指定位置。边上有个数字键盘，这样你不必把手挪开就可以做计算。我想打造一个平视显示器，就像战斗机那样，你可以直接看到数字显示在自己面前。1978年春，我用BASIC做了一个原型。

---

① 全称National Cash Register，其自动柜员机（ATM）、零售系统、Teradata 数据仓库和IT服务为客户提供关系技术解决方案。

采访者：有这个想法的时候，你有没有想到它会取得这么大的成功？

布兰克林：这个程序经过反复演变才最终成为VisiCalc。我把它介绍给自己在哈佛的朋友约翰·里斯（John Reese）他给了我很多鼓励。其次我想到要用一台Z80机器或类似的机器，以及一个电视屏幕。我觉得有个鼠标也不错。但是，当我开始在哈佛的机器上开发原型时，那台机器没有鼠标，我必须想办法确定位置。我要面对诸如"怎么取得这些值并进行计算，或者取得这些值并对它们求和"这一类的问题，因为哈佛的机器上没有鼠标。于是我想到了用行和列的方式实现这个功能。

随后，我找到我的产品学教授讲述自己的想法。他给了我很多鼓励。他对我说："知道吗，你提到的这类活儿，现在人们在做产品规划时都是在黑板上完成的？有时他们用的黑板拼起来有两个房间那么长。他们坐在那里，制定每周计划，销售多少，生产多少，库存剩多少。你的程序听起来不错。"我又找到会计学教授吉姆·卡什（Jim Cash）交流，他鼓励我说："商业产品之命脉在于良好的人机界面。这是现在许多系统设计都存在的大问题。"我觉得这家伙说得非常在理。



然后我去找我的金融学教授，他很令人沮丧。他从自己的打印资料里抬起头说："哦，已经有现成的财务分析系统，人们不会再买一台微机，比方说，去做不动产。你去找菲斯特拉（Fylstra）问问，他是我学生。他刚做完一项调研，他可以告诉你人们为什么不会买微机。"于是，我打电话给菲斯特拉，问他在做什么，他说："嘿，我跟未婚妻正在编写软件，并在自家公寓销售。我们正准备出售棋类程序。如果你有什么好玩的东西，不妨拿过来让我看看。"直到那年秋天，我才再次跟他联络。

那年夏天，在马萨葡萄园岛的骑行途中，我做了个决定，毕业后，我准备自己创业，力争把这个产品做好。如有必要，我会挨家挨户上门推销。那年秋天，我最后还是找到菲斯特拉，看看他有什么机器可用。他有一台Apple II和一台Radio Shack电脑。他答应把Apple II借给我，要是我想用它来开发点东西的话。

于是，我用一个周末写了个BASIC程序，电子表格只支持单屏，并设计了行-列、A-B-C和1-2-3坐标的操作方式。我还是想用鼠标，不过我设法用可以移动光标的遥控杆来替代。但是游戏遥控杆只能横向或纵向一个方向起作用。因此你得按下开火键，让它在横向和纵向之间切换。但是遥控杆用

BASIC控制反应太慢，于是我转而使用箭头键。由于Apple II只有两个方向键，我便用空格键代替开火键，用于切换横向和纵向。我不喜欢换挡键（shift key），我想尽量减少换挡键的使用，这也是我使用斜杠作为命令开始的原因，因为它不用换挡，手指也不必离开主键盘区，最大限度地减少了击键次数。

于是我改写演示程序，让它支持在屏幕上移动光标。你可以输入数字和公式，甚至可以点在公式里，通过指到A1上执行“1加7和1加A1”，就像现在的电子表格一样。只不过它没法上下左右滚动，只在当前屏幕内有效。重新计算大概要花20秒钟，每到一个单元就会发出声响，这样你就能听到它在计算。结果一半时间用在发出声响上。但这个演示程序让你体验到自己可以做什么。

我向几个同班同学展示了这个程序，其中有约翰·里斯，他指出，在引用另一个单元格时，你不必非得命令它使用那个单元格。我当时正在考虑怎么解析它。如果你要“1加”，然后敲箭头键，你肯定是想让1加上那个单元格。于是我反复试验这种方法，尽可能减少每个操作的击键次数。如今它已经成为我们所熟知的标准的简易电子表格、VisiCalc式界面。

我拿给另一位教授芭芭拉·杰克逊（Barbara Jackson）看。她评价说：“瞧，如果你想让公司董事长用它来做事，它就得非常简单。这很接近了，不过还是欠点火候。”正是这句话激励我把它做得越来越简单。

一两年后，当这款产品上市时，我演示给她看，对她说：“既然你是哈佛商学院计算器委员会的成员，就应该认识到这个程序的重要性，因为你的学生现在已经在使用它了。你必须做好准备。”实际上，那年商学教授就用它来编写考试答卷，并且吓到了其他教授，因为他们完成这些工作是如此之快。当然，现在哈佛大学会要求你在上他们的商学院之前先置办一台个人电脑。

一旦我们完成了人机界面，就该开始实际的编程了。在那之前，我就一直在构思数据结构，费了很多心思，琢磨怎么把它做得尽可能小巧紧凑，因为我们想让它在小的机器上也能跑起来。那时候，大部分Apple II只有16K字节内存，软盘并不常见。

采访者：弗兰克斯顿、你和菲斯特拉就是那个时候决定合伙的吗？

布兰克林：基本上，是的。我们就合伙事宜达成协定。弗兰克斯顿和我负责编写程序，菲斯特拉和他的公司（Personal Software）负责销售。我们会先在Apple电脑上开发这个程序，因为菲斯特拉刚好有台Apple电脑可供开发利用，并认为从这种机器下手是最好的选择。另外我们还有一些可以在Apple



电脑上使用的工具。

采访者：你是如何真正着手设计VisiCalc的？

布兰克林：我设计了内部结构，大量的数据结构和布局。在我看来，写程序最重要的部分是设计数据结构。另外，你还必须知道人机界面会是什么样的。因此我设计了非常紧凑的数据结构，足以容纳大量数据，而且访问速度很快。

我们还必须决定如何开展业务。当时我还在上商学院，鲍勃则在互动数据公司（Interactive Data Corporation）从事咨询工作。最后，我们决定从一个大型分时系统租用时间。幸运的是，他们没有要求一次支付几个月的租金，我们因此得以挺过那段时期。

鲍勃到晚上才写代码，因为晚上机时比较便宜。他下午3点左右起床，那时我会从学校回来。我们会仔细检查一遍他写的代码。我会测试程序、考虑如何添加新功能、跟会计师见面，并且处理创办公司所必需的其他工作。有时我会在那里待到深夜11点。下午6点资费会下调，晚上11点再次下调，到凌晨一两点，机器会变得快些。鲍勃一直工作到早上，然后回去睡觉。这个软件就是这么写出来的。



采访者：你们担心其他公司也会提出类似的想法吗？你们工作如此努力的动力何在？

布兰克林：一旦有了这么清晰的想法，你就想把它做出来。我们最担心德州仪器公司（Texas Instruments）会察觉到这个想法，并把它加进他们新推的计算机。苹果公司和Atari在签订保密协议后看过这款产品。Atari公司非常期待能得到一款产品，但是他们还没推出机器。我们拿到一台外观看似Cromemco机器的早期原型机，不过他们已经用Atari主板换掉了原来的主板。它是台实实在在的原型机。

苹果公司不怎么看好这款产品。那次会议我没有参加，我们的发行人菲斯特拉给他们带了一份VisiCalc早期原型的拷贝。作为程序员，我们认为4周内就能搞定。两三个礼拜后，鲍勃完成了大量功能，支持上下左右滚动，并实现了加减和重新计算。于是菲斯特拉又把那个版本拿给他们看。我则一直在优化BASIC原型，并尝试各种新功能。由于只有16KB内存，许多想法不得不放弃了。

在西海岸计算机展上，这款产品反响颇佳。我又找了几位教授帮忙。有

另一位产品学教授，他说的一番话非常振奋人心：“孩子，你看，完成所有这些计算只要一眨眼的功夫，真是太棒了。要是手工计算的话，那得用上几个小时。看来我得更新我书里的例子了。”

于是我写了一篇早期手册，鲍勃则设法实现我写的内容。我们精诚协作，只要鲍勃稍有懈怠，我就会说：“鲍勃，你得让这东西跑得快一点，不然不对劲儿。”或者我会说：“你得加上这个功能。”每当我说：“嗯，我们不如抄近路把这块绕开算了。”他则会说：“不，不，不，我们还是把它加上。”

他最后没有照搬我设计的单元格存储方式，而是做了一定调整。我做了大量设计工作，但内部的程序结构全是他做的。整个VisiCalc程序最终就是这么写出来的。

采访者：产品完工后，一开始的反响怎么样？

布兰克林：它得到零星的好评，但大多数杂志都没理会。大概一年后，有几本杂志才开始介绍。*BYTE*杂志登了一小段相关社论，仅此而已。这篇社论的主笔是菲斯特拉婚礼上的伴郎卡尔·荷尔默（Carl Helmer），他对这款产品有所耳闻。即使那一小段也是面向工程师的，写的还是我们尚未实现的正弦和余弦，就因为那篇文章提到了，我们不得不实现。我用了一个夏天才实现正弦和余弦以及其他类似功能。



我们借钱买了自己的分时系统。我卖掉一座房子，用那些钱交首付款。鲍勃则掏空了自己的积蓄，还找家人借了一笔钱。我们买了一台小型机，付了首付款，正式跨入商业领域。

采访者：当时你们决定怎么推广VisiCalc？

布兰克林：我们不清楚这款产品会卖得怎么样。我们知道产品非常好，每个人都应该买一套，但是当时还不清楚大家是否会争先恐后地用电子表格，因为他们还必须得买这台古怪的机器[1]，此外字处理器也没流行起来。

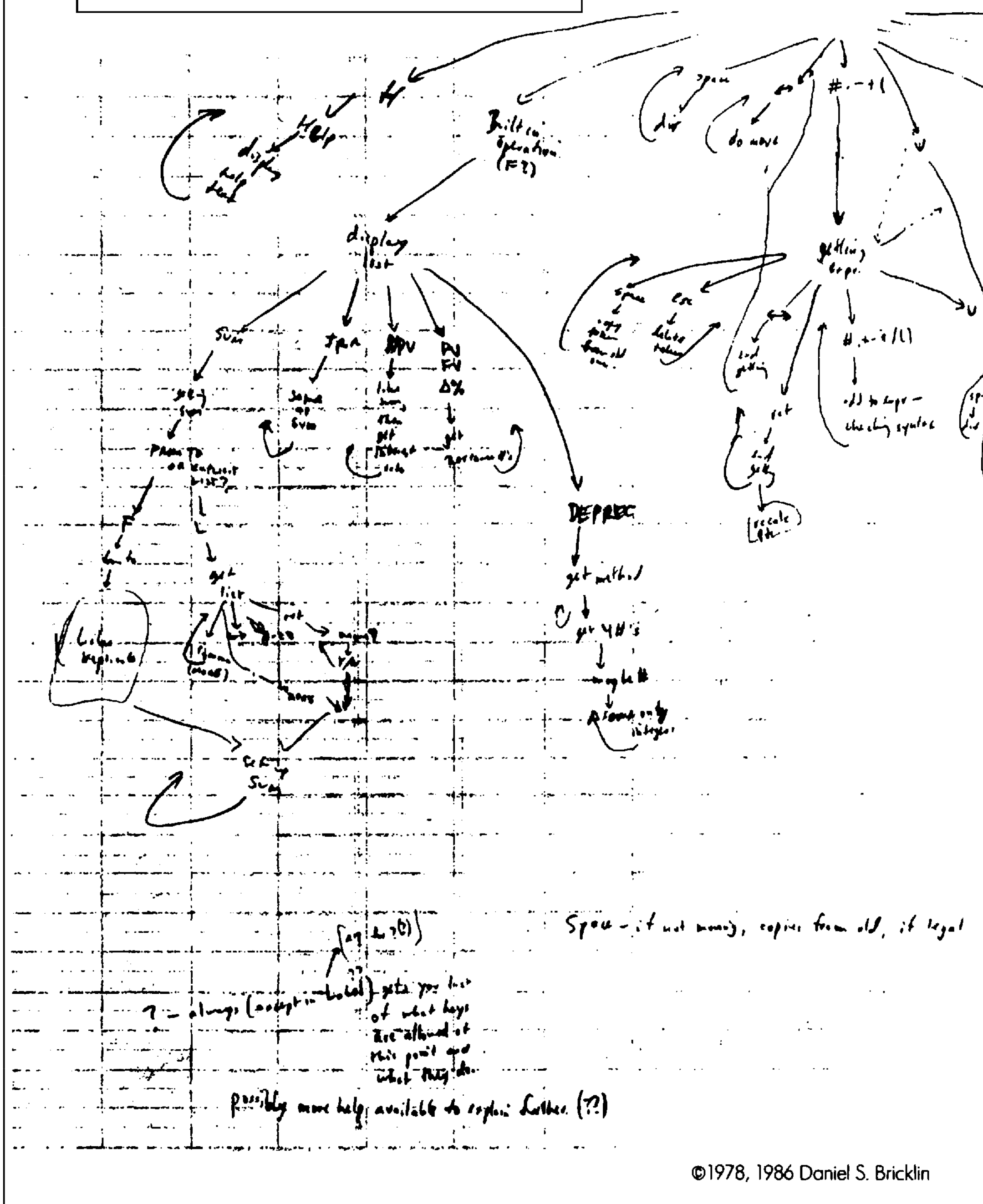
采访者：那么开办自己的公司之后，你又得做什么？你们跑遍全国各地去做演示了吗？

布兰克林：我们的发行商Personal Software在这方面做了大量工作。我们在全国计算机会议上做了展示，并正式宣布推出这款产品。我们还在西海岸计算机展和各种展会上进行展示。

---

① 指Apple II计算机，在VisiCalc发布之前还不太流行。

1978年11月，布兰克林最初的VisiCalc命令图解。该图画在布兰克林在商学院用过的一张试算表上（因此程序里也有网格）。



随后我们回去加班加点，把它移植到各种各样的机器上。我们把它移到TRS-80上，这款机器非常重要，在我们看来，这台计算机排名第二。完成这项工作的程序员是加入公司的塞思·斯泰因伯格（Seth Steinberg）。斯蒂夫·劳伦斯（Steve Lawrence）是我们的第一个员工。他做的部分编程工作完善了VisiCalc，尤其是在数值部分和其他一些功能上。然后，他开始实现Atari和Commodore PET版本。

对于Z80的版本，赛思决定逐字逐行把6502代码转译成Z80代码。我们有个不错的分时系统，因此想要什么工具，他只管造就行了。于是他写了个打印程序，可以并排列出6502代码和Z80代码。几年后，我们在6502代码中发现的缺陷在Z80代码里同样存在，反之亦然。这次转译做得真不赖。那些代码使用了很长时间。经过部分机械转译和部分人工微调，又诞生了IBM版本。

我们在IBM个人电脑上快速推出产品。IBM在宣布其个人电脑时一并宣布了VisiCalc。除非VisiCalc完成的差不多并且足以发布，否则他们不会宣布推出，他们非常挑剔。我们无法使用标准的转换程序做到这一点，因为我们有自己的汇编器。我们的程序员戴维·列文（David Levin）对汇编器做了修改，来实现从Z80代码到8086或8088代码的转译，并且标记了这些问题。做这些转换非常讲究技巧，它比其他多数转换程序要高明得多。正因为如此，我们才得以迅速推出IBM版本。就在IBM推出个人电脑几周后，我们就发布了新产品，并卖出了大量拷贝。



采访者：你们是怎么跟IBM达成协议的？

布兰克林：IBM先找到我们，因为版权归我们所有。根据我们的合同，他们应该跟Personal Software接洽，所以我们安排他们去找Personal Software。实际上，我们跟IBM达成了三方协议。IBM打电话过来后，我们第二天就跟他们会面。我们跟他们交代了合同的事，出示了有关哪些内容允许透露的合同部分，好让他们明白该怎么和我们打交道。对此我们非常坦率。

现在还记得，我们那时一直在跟许多厂商打交道，不过，对于那些成功和没成功的公司，我们都是全力以赴。因为你永远不知道哪些公司会成功。

采访者：当时你觉得IBM会取得成功吗？

布兰克林：我们着实用了一段时间才搞明白这台机器。毫无疑问，IBM机器的设计很了不起。不过，使用8088，并采用全键盘、80×25屏幕等，这在当

时很常见，许多人都这么做，包括当时最好的硬件设计师。很显然，它设计得很棒，这也是为什么我们觉得自己能达成目标的原因。

我们跟Personal Software在合同上存在诸多分歧。例如，他们想要支持什么机器，我们就必须把VisiCalc移植到那些机器上。他们想毁约，但又不愿付出足够的代价。为此我们不得不设法将产品移植到尽可能多的机器上。

采访者：谈谈你对Software Arts大起大落的看法。你想过你们可能会走下坡路吗？

布兰克林：虽然条件越来越好，我还是住在同一种房子里，换作在DEC或其他地方工作我也会住的那种房子。我在消费上始终很节制，不然我可能就没条件做现在做的事。我从未想过要依赖这家公司。

这就是我们为什么要在巅峰时期卖掉这家公司的原因，可惜我们最后惹上了官司。一旦官司缠身，公司就卖不了好价钱。我们的公司估值很高。在名义上，鲍勃和我都很富有，但尽管我们尽力了，仍回天乏术。依靠在业界的知名度，我有了一些可以带走的“资产”。这让我有机会重新创业。



从这家公司获得的经验很有价值。过去一年里，有几家公司提供过相当不错的工作机会，我都一一婉拒了。我借此明白只要愿意，到别的地方找份工作不成问题，但是我喜欢自己创业。

回顾过往，我非常满意。要是我们能一直保持巅峰状态，变得非常富有，那就太棒了，但是我们没有。不过换个角度看，当初我要是留在DEC，肯定做得没有现在好。

我喜欢在Software Arts工作。除了公司出现问题的那段时间，我在那里一直很快乐。即使是那段经历也让我学会了很多。在企业生命周期方面，我收获良多。经历这一整个周期，我对创办公司也更加自信。亲身经历了拍卖公司的椅子、终端、墙上的挂画、我的办公桌甚至我身上的T恤衫之后，我现在明白，尽管会有起起落落，人总可以挺过去的。

采访者：回顾Software Arts，换作现在，哪些事情你会采取不一样的做法？

布兰克林：嗯，要是没被起诉就好了。那样的话，情形就会大不一样。

采访者：具体是怎么回事？

布兰克林：我们收到一封邮件通知：“你们被起诉了。”多年来，我们一直和发行商VisiCorp（Personal Software）存有分歧，但那时我们已经就和解方案

谈判了一两年。不知出于什么原因，他们决定诉诸法律。也许他们认定自己握有对自身有利的证据，并会从他们的产品VisionCalc挣到钱。但他们没有事先核查一下VisiOn能否大卖。他们还没过河就把桥给拆了。我们没料到他们当真会起诉我们。

要是当时就有现在的认识，有些事情我可以采取不同的做法。如果市场早就起了变化，有些事情可以采用不同的做法。要是一些大型出版商如麦格劳希尔（McGraw-Hill）或其他公司愿意进入这一行？要是我们能说服苹果公司搭售这款软件呢？天知道。但是结果并没有那么糟糕，我又何必希望改变过去呢？改变做法的话，我就不会经历这一切。我仍然很健康，并不穷困，同事也都尊重我。除了没有变成富翁和我的公司历史不是完美无缺之外，从其他各个方面来看，我几乎都取得了成功。大部分人都会经历起起伏伏：看看苹果公司发生的事。

采访者：再给你一次机会的话，你会继续这么做吗？



布兰克林：我只做自认为是正确的事。我喜欢做现在做的事。我管理过大型软件公司，知道它是什么样的。现在，在小公司，我一样可以做自己想做的事。有时候会很棘手，不过这也是挑战。人生总是面临各种挑战：过去是创办一家大公司，眼下的挑战是要在一家很小的公司里取得成功。

采访者：你是否认为Software Garden源自你创办Software Arts的经历？

布兰克林：既是又不是。在Software Arts，我们必须试验很多想法：大公司，环境优越，还有研发机构。我认为，你要么能从产品上获得大笔收入，要么就得维持非常低的开销。还有我想试试不同类型的产品。我打造过不同类型的产品，我还想在不同的领域尝试更多。我想弄个小花园，而不是全力投入某一领域开办大农场，除非有一款产品能够大卖。

我现在通过邮递售卖产品，另外还通过分销商，他们会先从我这儿购买。我已经跟一家分销商签约，正在设法多找几家，以74.95美元的低价销售产品，并试着做些宣传。

采访者：你的新产品叫什么名字？

布兰克林：丹·布兰克林的演示程序。名字又好又简单，对吧？我决定叫它丹·布兰克林什么的，因为这么做有助于推动销售，很多人都这么跟我说。用户一看名字就知道软件的功能，它是做演示用的。

包装其实很简单。看到这个特卫强信封[①]了吗？说明书是用打字机打的，但效果很好。我决定在家运营业务，现在暂时就是这样。磁盘卡在一张硬纸板上，并装到信封里。包装就这么简单。

我知道怎么制作花哨的包装，但是作为用户，又会怎么处理这些花哨的包装呢？放在书架上又太占地方。程序的归宿是计算机，所以我决定化繁为简，何苦为难自己做不喜欢做的事？

采访者：你认为哪些人会购买这个程序？

布兰克林：程序设计人员，还有程序开发人员。文档编制人员会用它编写演示和教程。只要试用过几次这个程序，你就会发现它会帮你节省不少时间，值得购买。

另外，我也想看看当前做其他业务的可行性。如果你的成本尽量低，产品优良，知名度高，价格低廉，究竟能卖出多少份拷贝？有些人在这方面做得相当不错。我想一探究竟，因为它会对Software Garden出品的其他软件产生影响。



采访者：你觉得自己会一直从事软件编程吗？

布兰克林：我想是的，我似乎天生就有编程的本领。我不知道随着技术的发展它会变成什么样。也许我的技能更适合字符界面的屏幕，我在这方面的经验最丰富；图形界面则不一样，我在图形方面没有太多经验。也许技术会把我抛在身后，也许不会。

采访者：你认为这类小型软件公司还会涌现出来吗？或者，软件行业会由少数几家大公司主导吗？

布兰克林：不会。人们正在自己编写程序。使用电子表格的用户就在自己写程序，只不过现在用的语言不一样。

弗兰克斯顿喜欢讲一个有关电话公司的故事。20年代，他们认为电话数量增长如此之快，到了1950年全国上下人人都得成为接线员。到了1950年，人们都说："哈，他们错了。"好吧，其实他们说的没错，因为的确每个人都成了接线员，只不过他们直接用上了拨号电话。技术进步使得当接线员非常容易。

---

① Tyvek envelope，采用杜邦特卫强防护材料制作的信封。

其实编程也差不多。我们正在让用户自己做越来越多的编程工作，只不过他们不知道罢了。在Microsoft Word中使用不同的样式表就是在编程，使用电子表格也是在编程。

本质上，编程包含家庭手工业的成分。我们持续不断地改进可利用的工具、操作系统、环境和语言等。因此，个体总是能比他过去做更多的事情。

这就像写作。百科全书需要大量人工，一个人根本做不了。时下有些操作系统就像是百科全书。但是也总还有短篇故事、小说和诗歌。这些也自有其用途，尽管它们看起来简单，人们似乎不屑于去做。

大公司可以雇上100万个程序员，基于某个想法开发出更好的程序，但他们没有这么做，因为事情没有这么简单。经济学在这里不管用。有时，一个程序背后的想法就是小小的创造性工作。这就像短篇故事，情节上一个小小的转折蕴藏着整个构思。程序也是如此。

在市场营销方面，你的确需要一个庞大的营销团体。不过做市场也有很多途径。以图书销售为例，你可以自行出版，也可以找出版社。拿电影来说，你可以找发行商来做这项工作。



仔细观察软件行业销量最大的软件，你会发现，这些软件一般都是由几个人开发的。1-2-3其实是两个人编写和设计的，一个全职程序员，一个兼职程序员；VisiCalc也一样。我没记错的话，dBASE最初全是一个人写的。Paradox是两个人开发的。软件就是这么开发出来的。

采访者：*你认为软件会进入全新领域，还是会停留在原来的领域，比如电子表格、字处理器上？*

布兰克林：现在大家都在开发各种各样的产品。人们说："联网怎么样，我们为什么不能把这东西连接在一起呢？"出版系统呢，比如Mac计算机提供的廉价系统？还有Apollo和Sun等更大的机器上提供的庞大系统？几年前，没有人想到内部出版会成为计算机的主要用途之一。

人们总想让事情进展得快上加快。鼠标用了多久才投入商用？15年，抑或更久？恩格尔巴特（Englebart）早在60年代或70年代初就发明了鼠标。还有编译器，FORTRAN语言早在50年代就开发出来了。我们把那些事情当作理所当然，但其实它们是历经若干个5年、10年才做成的。而软件却每隔5年或10年就会有重大进展。那么，为什么要变得这么快？

一旦拥有特定的硬件，你就拥有了某些之前不具备的能力。而结果往往

是不用特定硬件就能做到某件事。例如，IBM计算机上之所以推出某些系统，是因为Macintosh上有，而且人们需要它们。Macintosh上的这类新硬件表明这些系统是有用的。因此，设法在旧硬件上实现它们也是值得的。

另一个很好的例子是Apple II。在它刚推出那阵子，人们自以为已经把它用到了极致。不过他们不断发现Apple II可以做更多事情。这就像试飞员玩命地试飞新飞机，甚至超出其预期设计能力。许多人因此坠落和丧命。在Macintosh上也是同样的道理。

采访者：*你认为Macintosh存在哪些不足？*

布兰克林：Macintosh里有一些特性会像塞壬海妖一样引诱你触礁，比如使用字体。使用不同字体会大幅减慢字处理器的响应速度。结果字处理器会慢得像蜗牛爬，令人无法忍受。人们喜欢速度快的东西。一旦你使用了不同字号的字体，系统就会越来越慢，尽管最终效果看起来很漂亮。无论怎样，最后你对产品的感觉并不怎么样，因为你听从了海妖的诱惑了。最终你认识到要避开这种用法。



采访者：*速度、用户友好或其他类似事项到底有多重要？*

布兰克林：对部分产品来说，这就是产品的全部。产品本身就是它们造就的。

假设你在大力神图形卡上运行Microsoft Word。要是他们没有对它做大量增强工作，有些功能的运行速度会变得非常慢，比它原本运行的速度还要慢。然后，有些人就会无法忍受Word。尽管它在功能上完全一样，但潜在用户数量也会大幅减少。

我认为产品体验非常重要。Lotus的成功便是印证。有一款叫Context MBA的产品，同样非常强大，功能丰富，但它的速度如此之慢，让许多人从未想过要试试这款产品，因此它怎么也流行不起来。而Lotus则非常注重速度。

对于VisiCalc，我们实现了跟原来Apple II相同的速度。我对鲍勃说："这还不够快，我们得让它更快些。它的滚动速度必须跟你机器上的重复键一样快。"我不知道不同的机器重复的速率并不相同。我们采用的大概是中等速度。我们试图在IBM计算机上做出同样的效果，最后感觉不错，非常清爽。

采访者：你认为编程是技能还是科学？

布兰克林：部分是科学，部分是技能。科学里描述的某些内容也可能很有用，比如电源设计等。但编程也有手艺的成分，见习训练是编程的必经之路。通过见习训练，你可以掌握技能。有些事情，说到底就是熟能生巧。因此，它更像是一门艺术或手艺。编程和当精算师不同，它更多的是标准化的手艺。

从开发人员的某些程序中，你可以感觉到这点，这感觉会在你看他们的代码时自然而然地浮现。受过正规训练的人往往比没受过训练的更具优势。一般来说，知道人们在学业上取得过什么成绩会很有帮助。你看，有些人有园艺技能，有些人则没有，但要是事先学过一些这方面的知识，肯定有帮助。

采访者：每次编程时，你基本上都会采用同样的方式或做法吗？



布兰克林：不会。这取决于我做的是什么。我的意思是，我会尽量做到专精，但不是每次都一定要这么做。不过，如果你开发的是商用软件，又想把东西做出来，那就不能太过死板，比如“我只用高级语言编程”或“真正的程序员非汇编语言不用”。你只管怎么合适就怎么做。

采访者：你认为5年或10年后软件或电脑会扮演什么样的角色？

布兰克林：我认为未来的电脑会比现有的形式更加多样化。一种形式就是放在桌面上的箱体，也即今天的个人电脑。

我觉得将来的个人电脑会跟台式机有所不同，现在的电脑就像是放在你办公桌上的电话。台式机之所以没在家庭用户中流行起来，是因为我们今天所知的个人电脑对家庭而言就像大型机之于企业——你得步入圣地，也就是进机房联机。很多人都不想连接到大型机，把卡片塞进去，并做些诸如此类的事情。人们也不会为了查看食谱而专门走进小机房。那太搞笑了。

未来的个人电脑应该更像是个记事本。我的记事本随身携带，为什么它就不能是电脑呢？嗯，这不同于我们现在熟知的个人电脑。计算机技术会用在各种各样的全新场合，比如用作记事本。

假设语音识别技术一直发展下去。你身边有一台能理解命令的小录音机，那么为何还要按按钮呢？你只消说：“播放磁带末尾5分钟。”它不用连到台式机上就能做到这点。这会成为录音机自带的功能。

你现在用的台式机也会日趋完善。价格是一个重要因素。以复印机为例，现在新机器的价格跟过去购买旧机器的价钱差不多，但是功能已经有所提升。我认为个人电脑将出现多种不同的形态。

波西娅·艾萨克森（Portia Isaacson）[①]曾说，她认为机器人会变得举足轻重，因为你的电脑应该紧随你左右。要让电脑伴随你左右，有一种方法是使之小型化。如果我们能把电脑做得小到可以随身携带，何苦还要给它安上腿呢。用不了多久，我们就能做到在很小的空间里提供强劲的运算能力。

续写传奇人生

布兰克林在1985年的时候二次创业，建立了Software Garden。有了之前创业的经验，布兰克林有意让新公司保持较小的规模。Software Garden是一家软件顾问和开发公司，主要产品有“丹·布兰克林的演示程序”，以帮助用户在编写程序前创建该程序的演示，也可用来为基于Windows的软件建立说明手册。布兰克林一直担任Software Garden总裁，直到1990年退出该公司，与他人共同创立Slate公司为止。Slate公司主要是为手写电脑和移动电脑开发软件。1992年，布兰克林开始担任该公司副总裁。然而，因为这种计算机销售不佳，Slate也于1994年关门大吉，于是布兰克林重新回到Software Garden。

1995年，布兰克林创立Trellix公司，该公司于2003年被Interland（现在的Web.com）收购，此后布兰克林一直担任Interland的CTO，直到2004年。

2002年8月，布兰克林提出了“friend-to-friend网络”的概念，这是一种P2P网络，用户只能和自己认识的人建立直接连接。同时，他还提出了“逆公有”（Inverse commons）的说法，即个人会为社区的发展贡献知识和内容，而非为个人目的获取内容，典型的例子有开源软件和维基百科，这种现象此前亦被称为“公有之喜”（comedy of the commons）和“公有聚宝盆”（the cornucopia of the commons）。

布兰克林目前依然是Software Garden的总裁，销售他自己开发的程序，同时也提供演讲和顾问服务。Software Garden的网站上列有Notetaker HD（iOS笔记软件，整合手写功能）、wikiCalc（在线协作电子表格）等多种产品。



① 生于1942年，计算机领域的先驱，1986年提出全自动智能住宅的原型。

The Official

SAAS

Gift Catalog

Special Summer Edition – Volume 1

Durable Hot & Cold Mugs



Made of high grade organic styrene and capable of holding most reasonable liquids. Not available in any store.

ORDER #3623 – MUGS

SPECIAL – LIMITED OFFER



Remember that lovely cactus in Dan Bricklin's office? Well, we bought that cactus and for a limited time only are selling clones — each in its own pot and genetically certified as a direct descendant of the original Bricklin office cactus.

ORDER #5984 – CACTUS

(We cannot ship this order to the Soviet Union or to any Soviet bloc country because of technology transfer restrictions.)

Start Your Own Theater



We have hundreds of spotlights. We have no idea of what they were used for or even if they were used at all. Most of them are almost new. All spotlights come with bulbs and cables.

ORDER #7124 – LITE

LUCKY GRAB BAGS !?!?!?!?

Each Bag Contains One Pound of Stuff

675 Mass. Ave Phone Lists / Central Square Parking stickers / β-test floppy disks / Mica Lane card key cards / Diet Pepsi empties / video-tape bloopers and outtakes / Federal Express receipts / telephone credit cards and tons more ... Everything must go!

FREE: Small box of β's, one cubby spud and a screenful of electronic mail with each grab bag.



ORDER – 9887 – BAG

Specify how many.

此页来自“礼品目录”，是Software Arts公司的一个员工为公司倒闭时举行的拍卖会而写的。



9

鲍勃·弗兰克斯顿

36岁的鲍勃·弗兰克斯顿（Bob Frankston）从事编程工作已经超过20年了。他在纽约的布鲁克林区长大。早在少年时期，他就对电子技术和计算机产生了浓厚的兴趣。在麻省理工学院上学时，他深入钻研了这些领域，并于1970年从麻省理工学院获得了两个学士学位，一个是数学学士，一个是电子工程和计算机科学学士。1974年，他又从麻省理工学院获得了另外两个学位，一个是工程学士学位，一个是电子工程和计算机科学硕士学位。他在麻省理工上学时遇到了丹·布兰克林，两人成了朋友。

布兰克林在哈佛商学院的时候产生了电子报表的想法，于是请鲍勃·弗兰克斯顿帮助开发出一个可运行的版本。弗兰克斯顿不管白天黑夜，只要一有时间，就会在他的小阁楼里编程，最后开发出了一个以布兰克林的构想为原型的电子报表的软件版本。后来他们两人成立了Software Arts公司，并肩工作，于1979年开发出他们的第一个产品——VisiCalc。1985年春天，公司因为与 VisiCorp公司（以前的Personal Software公司，后来的Paladin公司）打了一场旷日持久的官司而解散，弗兰克斯顿随即加入了莲花软件开发公司，成为信息服务部门的首席科学家。

莲花公司的新办公大楼坐落在坎布里奇镇的坎布里奇街上，隔查尔斯河与波士顿遥遥相望。我到那里的那天，天空正下着倾盆大雨，空旷的三层楼高的大堂上方，新安装的玻璃顶蓬有几处正漏着雨。



一个留着棕色长发、戴黑边眼镜的小个子拿着一把雨伞走进了大堂，他就是弗兰克斯顿。他气喘吁吁、浑身湿漉漉地过来和我握手。然后，我们搭电梯上楼到了他还没有来得及搬进去的新办公室。我们穿过了一个有很多小隔断的大房间，我发现有很多隔断是空的，不知道是因为隔断比人多，还是因为有很多人还没有搬进去。

弗兰克斯顿的大办公室位于大楼的一角，有一面正对着河。办公室里有两张大木书桌和一张圆桌，但没有椅子。他从旁边的会议室里拖来两把椅子，然后又冲出去带回两杯热咖啡。他坐下来，从包里拿出一个纸袋，袋子里是刚出炉的新鲜玉米松饼，于是我们开始边吃边聊。

从弗兰克斯顿的言谈举止上看，我觉得他是一个习惯于忙忙碌碌的人。他语速很快，言词犀利，思维跳跃，虽然话语不多但机智活泼，有着一股严肃认真劲儿。他觉得自己是一名工程方面的创新者，但他让我觉得像是一位老师，总是试图用与听众有关的事情来说明问题。在谈话过程中，他丢给我的问题并不比我问他的问题少。



* * *

采访者：你是怎样进入计算机软件这一行的？

弗兰克斯顿：1963年，我还是一名初中生时，就坐在纽约一所大学的课堂上学习使用IBM 1620系统了。我发现这种经历特别能发挥我的创造力。我感觉可以借此改变世界，并且我很喜欢编程的灵活性。

采访者：你觉得编程是艺术、科学、工具还是手艺？

弗兰克斯顿：所有这些都是。当我们成立公司开发VisiCalc的时候，我们把公司命名为“软件艺术”（Software Arts）。或许应该这么问：艺术和工程的区别是什么？

采访者：那二者的差别是什么呢？

弗兰克斯顿：差别并不大。在工程方面，如果做错了，会很容易看出来，因为有些东西会出问题。但好的工程和好的艺术并没有差别。它易于理解吗？便于维护吗？是否简单？是否有不必要的复杂性？如何理解它？在这个意义上，艺术和工程是相似的。当然，在纯艺术领域，很少需要操作测试，所以出点问题也不容易发现。工程领域比较实际，因为需要让某些东西可以工作。但它是不是工作一次就散架了？是不是为了修复一个bug会产生出更多

的bug？一般来说，越是优秀的工程师，生产的机器就越有艺术的美感，也越有可能正常运转。

采访者：你觉得计算机程序的哪方面最具美感？

弗兰克斯顿：我想不光是一个方面，而是很多元素的组合。程序是否易于理解？是否优雅？是否体现出对目标的基本理解和精巧的设计，还是只是一些元素杂乱无章地堆砌在一起？是否容易修改、开发或变更？

采访者：VisiCalc和Software Arts是你取得的第一个成功。这个产品和公司是怎么来的？

弗兰克斯顿：那时我和丹·布兰克林正在编写VisiCalc，我们觉得成立一家公司来保护我们的商标比用合伙人的方式要好。我记得我们在一家餐馆里（我开玩笑地叫它KFF肯塔基炸鱼店）想出了“软件艺术”这个名称。公司的办公室就在我家阁楼上。丹那时还是全日制学生，所以我就负责编程，而他主要做的是设计和评估。



采访者：你和丹·布兰克林为什么会把宝押在VisiCalc上呢？以前从来没有人做过这样的事。

弗兰克斯顿：我们有很多相关背景。丹以前在数字设备公司（DEC）工作，负责DEC字处理器，那个软件他们到现在还在卖，叫做DecMate II。因此他具备屏幕界面的经验。我们最初的想法是把一台电视和一个手持式计算器集成起来，遥控屏幕上的东西。

采访者：从最初的电视想法发展到最终的产品花了多长时间？

弗兰克斯顿：几个月。从我开始编程到我们第一次做演示大约花了4个星期。然后又花了10个月的时间才交付成品。但我们并没有把所有的时间都花在编程上，我们还在创办公司。

采访者：对于新闻舆论对VisiCalc的反应，你是否感到惊奇？

弗兰克斯顿：成功来得并没有那么快，大概用了两年时间。人们对VisiCalc的认知非常慢，这让我们有些失望。实际上，很多人到现在都还不了解。例如，他们没有把它当成是一种编程语言，他们并不明白这方面的意义。

采访者：你在开发VisiCalc的时候，是否意识到这将会是一个巨大的成功？

弗兰克斯顿：没有。我们只是觉得它会是一种很有用的程序，比干一份拿薪

水的工作好。更主要的是，做一件大家会用到的东西是件很有意思的事，就好比造一台新机器。一旦我们要让大家使用这个产品，需要考虑的就不只是编程了，还要考虑美感和可用性。如果计算机是有用的，就要诱导人们把计算机真正用起来。

采访者：你认为电子表格、字处理器以及其他办公程序会一直变化吗？

弗兰克斯顿：当然。万事万物都是永远在变化的。人们直到现在还在使用象形文字。观念是不会消失的，它们改变的是形式，它们会与其他观点融合起来。

采访者：你现在还和丹·布兰克林见面吗？

弗兰克斯顿：是的。他住得离我并不远。我还和很多老朋友见面。丹觉得不用花太多费用、自己一个人坐在房间里编程比较有意思。而我则倾向于做一些需要更多资源的东西。



采访者：你认为计算机今后还会继续以目前的方式使用吗？

弗兰克斯顿：不会。总的说来，我认为现在的这种计算机会消失。计算机会变成智能代理或家用电器。个人计算机最终将销声匿迹。当然这个过程是逐渐的。有一个阶段会有很多针对个人计算机使用者的应用，但这决不会是进化的终点。虽然人们还无法准确预言计算机究竟会怎么发展，但谁会喜欢在桌子中间放这么一个大盒子呢？

采访者：当计算机因为发展而变得越来越容易使用时，软件会怎样发展？

弗兰克斯顿：人们会需要越来越多的软件，但是程序必须越来越容易编写，这也会让大多数程序员被淘汰，因为非专业程序员已经在编程了。VisiCalc作为一种编程语言就是一个例子。任何会操作电话的人都可以编程。

采访者：即使这样，目前大部分编程工作还是由一小部分人完成的。是什么因素决定一个人能成为一名优秀的程序员，而不是众多用户中的一员？

弗兰克斯顿：什么因素使人在某一方面出类拔萃？是什么因素让人成为优秀的作家？优秀的人一般需要满足两个因素：正好具备这个知识领域所需要的心智，并且在能力上又不是太愚钝。这样的组合很罕见，但并非不可思议。

很多编程技巧都是可以传授的。一个优秀的程序员必须喜欢编程，对它感兴趣，这样他才会努力多学一点。优秀的程序员还要对美学有感觉，并且有相应的负罪感，能够敏锐地意识到什么时候违反了程序的美感。负罪感迫

使他更加努力地去改进程序，使程序更加符合美感。

采访者：如果把编程和一种艺术形式相类比，你会把它比成写作、雕刻、绘画还是作曲？

弗兰克斯顿：在音乐和其他艺术形式中，我们会试图遵循一定的规范，但同时我们也会在某些方面突破这些规范。在音乐中有很多规则，你需要知道什么时候突破它们，什么时候遵循它们，就好像编程一样。

教人写作的时候，可以告诉他从列提纲开始，这样可以使文章更有组织和条理。在编程领域，有很多关于组织和条理方面的技术可以教。如果你是一名雕塑家，你最好懂得重心在哪里，或者至少对重心有点感觉，否则，当作品倒下来的时候是有可能砸死人的。

在艺术领域，你会问自己别人如何感知你的作品。你会尝试建立一种特定的感性印象。我认为很多艺术家对主题的看法差异会很大，就像许多程序员一样。在交流的时候，不管是写作还是编程，你所表达的内容必须能被对方理解。如果连你自己都无法把程序解释清楚，那么计算机能够正确运行的可能性就很小。



采访者：计算机科学中的科学成分占多少？

弗兰克斯顿：计算机科学这个词用得有点过度了。我更喜欢用软件工程、计算机工程或信息工程。因为工程没有那么绝对，所以我更愿意把一些事情看作工程。当然，编程是带有科学成分的。

我认为计算是一个比较窄的概念。很多科学主要是理解交互关系的复杂性。但对编程本身而言，我觉得更多是工程规范。

采访者：你在编程中采用的是什么样的工作策略？

弗兰克斯顿：对所做项目的进展状况有个总的认识是很重要的。从这里着手，我会仔细考虑设计构想，继而制定一个程序框架，作为后续工作的基础。然后，建立一个工作骨架来即时地满足设计需要，同时也可提供一个途径，在需要时可做相应的测试和加工。可以说程序是有组织的，而我促进了形成的过程。最后，项目会进入到一个阶段，所有的原则都已规划好，我要做的是确保所有的细节都考虑周全并顺利交货。

采访者：对你来说，现在编程比你20年前开始时要容易吗？

弗兰克斯顿：我认为是容易了，但我仍在试图做些更复杂的项目。有了现代

化的计算机，你发现超出自己能力的地方比以前多了。比如，我认为自己在学校里是一个拿分数B的学生。如果我对课程感兴趣，就会拿A，如果我不感兴趣，就会拿B，这都取决于我自己而不是科目的难易程度。同理，在编程时我会利用手头的工具去尽力而为。

采访者：在编程的某一个阶段，你会满脑子都在想整个程序吗？

弗兰克斯顿：我通常会总想着它。在不同的时候，有时想得多点儿，有时想得少点儿。在更多的时候，我会想着哪里可以更详细些，于是这儿加一点，那儿加一点。

采访者：你在程序中会写很多注释吗？

弗兰克斯顿：那要看我觉得代码是否易于理解。程序语言越高级，我写的注释就越少。我尽力编写可读性好的代码。在编程语言中，我只使用有意义的常规用语，如果我觉得一段代码不是很容易读懂，就会写上注释。

我曾听人说我的代码很容易接手，我写代码的确是为了让人很容易读懂和接手。另外，我这样写代码可以让我在修改程序时不必太担心影响效率。注释的主要目的是为了对异常发出警告。



采访者：你的程序构想是从哪里得到的？

弗兰克斯顿：从我想做的项目而来。这些需求往往与别人觉得什么是有用的东西相关，我需要和人对话。我不是在真空中运作。我喜欢与人合作创造出有用的工具来。我用计算机来表达我认为事情应该怎样做。

采访者：你目前所做的项目是什么？

弗兰克斯顿：嗯，是工具和产品的组合。我正在一个新的工作方向上。这不仅仅是编程和在老的问题领域中做产品。

采访者：对你来说，编程中最难的部分是什么？

弗兰克斯顿：这有很多不同的测量标准。在一般情况下，我尝试找到正确的观点或方向感，但是这不只是在编程上。这就像在写作，什么是适度的解释？什么是错误的观点？发挥得太多了吗？编程最困难的部分是使产品预备好可以交货，使产品可以工作并满足用户的需求。

采访者：当你带领很多程序员做一个项目时，会有很大的不同吗？动力是什么？

弗兰克斯顿：你尝试过与10位作家合写一篇的文章吗？那只是任何一个项目

中存在的正常动力。问题是，在做大型项目时，你需要能够与其他很多人一起合作，你不能总是临时决定设计和各种问题。你必须思考最终结果，给人们提供目标、设立规则，而不是一上来就说："这主意不好，不能那样做，要这样做。"你会发现如果这样做的次数太多，大多数人都不会有很好的回应。

采访者：你对年轻程序员的忠告是什么？

弗兰克斯顿：总的来说，不要以为你知道所有的东西，要尝试学习并质疑那些假定的东西。要信心十足，但要保持谦虚，要猜想你可能做错了什么。要有刚好够多的罪恶感——不要太多，不然就会害怕去做任何事情——但要足以建立美感。尝试去做更深刻的理解。不要因为你曾使它运行成功了，就以为再也没有什么需要去了解的了。

我认为当今程序员的一个问题是，当你能用BASIC来做些东西时，你以为你都明白了，其实你才刚刚起步。在编程中，我发现很多的理解来自心理学，来自人们是如何与这世界打交道的。最终，你要做的是试图了解一般的工作流程是怎样的。程序只是一小块。但是你需要有好奇心。你的编程可能只涉及很少的部分，但你最好有好奇心，并试着去学习新的理解方式。



采访者：你是怎么看待人工智能的？

弗兰克斯顿：问题是近来对人工智能的炒作有点太多了。如果明白了发生的事情，你就不愿意再使用智能这个词了。许多商业专家系统不过是决策表而已，过于庞杂而不易解决实际问题。它没什么神秘的。我会把人工智能看成是一个比一般系统在理解认识与思考上更有意思的系统。

因为人工智能是用来理解如何处理复杂问题的，所以我对人工智能充满热情。我们可以从人类的智能中学到很多东西，人类的智慧可以管理并完成壮举，而用到的也许只是一个平常的计算引擎。当我们试图了解人类的智能时，我们也就发展了处理复杂性的原则。公平地说，今天在计算机上所做的很多东西都是源于对人工智能的探索。

采访者：你觉得IBM个人计算机怎么样？

弗兰克斯顿：IBM PC很不错，现在这种计算机也很多。也许今后的几年中我还是会继续在IBM PC上编程，但我更愿意在大量存在并且我能够很好利用的计算机上编程。目前IBM PC有很多严重的限制。现在的问题是：我们得有多聪明才能够避开并摆脱这些限制呢？我觉得令人兴奋的是切实推动自己去实现在更大领域中的目标。编程本身不是目的。

续写传奇人生

与丹·布兰克林共同创建的Software Arts公司解散后，弗兰克斯顿于1985年加入莲花公司，在那里开发了Lotus Express产品，并为Lotus Notes设计了一种传真设备。1990年，弗兰克斯顿加入Slate公司，从事手写电脑和移动电脑的软件开发。1993，弗兰克斯顿进入微软公司。他在微软工作了5年，致力于研究计算机的消费应用，特别是家庭网络。1998年，弗兰克斯顿离开微软，加入CommonAngels，工作重心转向早期投资和种子投资。

近年来，弗兰克斯顿积极提倡在互联网进化过程中降低电信公司的作用，还创造了一个新词“Regulatorium”来形容电信公司和反对变革的管理者（Regulator）的勾结。



```
/*
Cubic spline fitting - Ellis-McLain Method
Jonathan Sachs
22-Oct-85

This method generates a single-valued function from an ordered set of x,y data.
It produces the best results when used on smooth functions since it reduces
discontinuities in the second derivative.

The findgrad procedure is called first to determine slopes at each data
point. This step may be omitted if slopes are already given for each data
point. Next the coeff procedure is called for each interval to generate the
coefficients of the cubic polynomial that fits the data in that interval.

Reference:

Ellis, T.M.R. and McLain, D.H. (1977) "Algorithm 514 - A new method of cubic
curve fitting using local data". ACM Transactions on Mathematical Software,
Volume 3, pages 175-178.
*/

/*  find gradient at each data point

    findgrad(x,y,grad,n)

    x        n-vector of x coordinates of data points
             values must be in increasing order with no two equal

    y        n-vector of y coordinates of data points

    grad returned n-vector

    n        number of data points (must be at least 4)

*/

findgrad(x,y,grad,n)
double x[];
double y[];
double grad[];
int n;
{
    int i,iless2,iless1,iplus1,iplus2;
    double x0,x1,x2,x3,x4,y2;
    double prod1,prod2,num,denom,g;
    double coeff2,xdiff,xprod,weight;
```

萨奇的这个小程序是为了从一个有序x，y数据对中产生一个单值函数。完整的程序清单请查看附录。



10

乔纳森·萨奇



乔纳森·萨奇（Jonathan Sachs）出生于1947年，在美国东海岸的新英格兰地区长大。他获得了麻省理工学院的数学学士学位。萨奇在麻省理工学院一共学习和工作了14年。作为程序员，他的编程经验很广泛：他曾在空间研究中心、认知信息处理小组以及生物医学工程中心工作过。在为生物医学工程中心工作时，他开发了STOIC（面向堆栈的交互式编译器）编程语言。

在70年代中期，萨奇离开麻省理工学院，来到Data General公司，负责管理一个操作系统的开发工作。接着他与人共同创办了Concentric Data Systems公司，一个以数据库产品而知名的公司。取得非凡成功的Lotus 1-2-3电子报表软件的开发工作要归功于乔纳森·萨奇。1981年，他与米切·卡波（Mitch Kapor）一起开发并推销萨奇的电子报表程序，在1982年4月，只有8名员工的莲花软件开发公司成立了。1983年1月26日，莲花公司开始交付用于IBM PC的Lotus 1-2-3软件。同年4月26日，Lotus 1-2-3软件首次在Softsel畅销排行榜上跃居首位，并从此居高不下。它也是第一个取代VisiCalc的程序。1984年，萨奇离开莲花软件开发公司并成立了自己的公司。

在一个秋季的大雨天，我前往马萨诸塞州坎布里奇的西边，乔纳森·萨奇告诉我在那里和他见面。在对建筑群办公室的索引图搜索一番之后，我来到了一扇简单地标着“乔纳森·萨奇联合公司”的门前，我预想的可比这儿

要气派得多。

我敲门进到一个大的单间办公室中，房间里播放着爵士乐，洁白的墙上挂着几幅安塞尔·亚当斯（Ansel Adams）的画。一个身材高大、头发略带花白、留着胡子、穿着羊毛衫和灯芯绒牛仔裤的男子从他的计算机终端前站起来，温和地跟我打招呼。这就是那个开发了Lotus 1-2-3软件的萨奇。我坐在一张休闲皮椅上，他关了IBM AT机，把椅子转了过来，开始和我交谈。

* * *

采访者：你是从上大学的时候开始编程的吗?

萨奇：是的，1964年，当我还是麻省理工学院一年级的新生时，我修了一门编程简介的课程。后来，我的好几份暑期工作都是编程——我曾在伍兹霍尔干了一个夏天，在喷气推进实验室又编了一个夏天的程序。我在大学最后一年辍学去赚钱的那段时间，也做过程序员。编程是我赚钱的一项技能，但开始的时候我对编程并不很着迷。



采访者：为什么不很着迷呢?

萨奇：一方面，最初我只做FORTRAN的批处理编程。我对FORTRAN没有什么不满意的，但每天只是反复地运行几次程序让人觉得很沮丧。我在大学最后一年辍学期间，为麻省理工学院的空间研究中心分析卫星数据。后来我去了旧金山，但过了3个月都没能找到工作，所以又回来继续完成学业了。在我回到麻省理工学院的第一天就得到了一份有趣的兼职工作，为认知信息处理小组编程。他们当时正在为盲人开发阅读机。我在一台PDP-9的微型计算机上工作，参与字符识别部分的开发，就是用扫描仪来读取页面上的信息。

那时我才真正对编程产生兴趣。那是第一份让我可以坐下来与计算机进行实际交流的编程工作。让计算机按照你所期待的方式做出响应，那种感觉令人兴奋。那次经历令人非常激动。我记得特别清楚。

采访者：你为什么决定回到麻省理工学院并攻读学位?

萨奇：对刚入行的人来说，没有学位很难找到工作。而且麻省理工学院是一个安全的地方，因为我以前曾在那里待过。

在获得学位后，我又回到空间研究中心。他们正计划发射一颗X射线卫星，希望有人能设置好一台小型计算机，对电话传来的部分数据进行分析。

我负责准备计算机和所有的系统编程。几位研究生写了一些分析程序，我完成了剩下的程序。

卫星发射后（那是一次巨大的成功），我在那里再也得不到搞计算机的机会了，所以我离开那里并在学院的生物医学工程中心找到了工作。他们正在研发一种以8080为内置芯片的医疗器械，需要有人为他们开发编程语言。于是我写了一个叫做STOIC的语言，那是FORTH语言的一种变体。

采访者：你为什么把它叫做STOIC？

萨奇：STOIC的意思是面向堆栈的交互式编译器（Stack-Oriented Interactive Complier）。我在白板上写满了所有能想到的首字母，希望找出最佳缩写，最后选择了STOIC。现在STOIC仍然存在，用于公共领域，偶尔还会被人引用。实际上，我认为爱普生公司（Epson）就使用STOIC开发了一些软件。

总之，在麻省理工学院一共待了14年之后，我觉得应当离开了。我希望进入管理层，因此加入了Data General公司，负责管理一个操作系统的开发。我在那里工作了大约两年半。那项工作使我明白自己并不是很喜欢管理工作，而且我再也不想开发什么操作系统了。对于我曾经抱怨过的人，我现在更多的是对他们的同情。



采访者：你是自己开发那个操作系统呢，还是管理一个团队的程序员？

萨奇：团队最多时有8个程序员。在我离开Data General公司一段时间后，我的上司约翰·亨德森（John Henderson）也离开了，后来我们两人成立了一个名为Concentric Data Systems的小型咨询公司。我们承包了一些软件编程工作，还为Data General公司的硬件编写了电子报表软件。但我们很快发现，虽然开发电子报表软件看上去是个不错的主意，但我们却赚不到什么钱。

这段经历动摇了我和亨德森的关系——他所做的外包工作是赚钱的，而我所做的产品开发工作是不赚钱的。我们面临的一个问题是，两名纯技术人员不应该去运作一个公司。我们在本应做市场营销的时候却在技术问题上争论不休。于是我们分开了，我拿到的是在旧电子报表软件上开发另外一个电子报表软件的权利。

我把电子报表软件拿给米切·卡波看，此前他的市场营销很成功。他卖了VisiPlot和VisiTrend个人软件，并得到版税。他知道用户需要什么，而我知道如何实现那些功能。我们的业务合作关系处理得很好。

采访者：在当时，你只是有了Lotus 1-2-3的想法，还是说软件已经完成了？

萨奇：当时电子报表程序已经完成了，我在一个月内将程序转化成了C语言版本。接着从那时开始，每次加入一点新功能。实际上，最初的想法与最终成型的Lotus 1-2-3软件版本有很大差别。

采访者：软件最初的想法是什么呢？

萨奇：要在其中构建更多的编程语言功能。但我们很快发现那几乎没有什么市场，还不如做单纯的电子报表软件。后来我们加入了宏语言，这样就又拾回了构建编程语言功能的部分想法。

Lotus 1-2-3软件原本打算包含电子报表软件、图形处理器和字处理器。但在开发过程中，我们看到了Context MBA软件的原型。其中的字处理器让整个软件陷入了僵局。所以我想，也许加入数据库功能会比加入字处理器好，而且数据库功能也更容易实现。



所有这些开发决定似乎都是经过慎重的市场调查后得出的结果，但实际上却不是那样。我们用于开发Lotus 1-2-3的方法与该产品的成功有着很大关系。举例来说，Lotus 1-2-3从一开始就是可以使用的软件，而在整个开发过程中，软件一直都是可以使用的。当时我几乎是单枪匹马一个人工作。我住在马萨诸塞州的霍普金顿，在那里有一个办公室，我大约每周去一次办公室，带去新的版本。对于出现的bug，我在下一版本中会立即修复。而那时，莲花公司的工作人员也一直在不间断地使用这一软件。

这种方式与开发大型程序的标准方式刚好相反。开发大型程序的标准方式是花费大量的时间编写功能规格说明书，做好模块分解，为每个模块分配一些程序员，当各个模块完成后集成在一起。这种方法存在的问题是直到很晚才能得到一个可以运行的软件。如果确切地知道要做什么，这种方法是不错的。但在做新东西时，会出现各种意想不到的问题。不管怎么样，我们的方法都意味着到了软件开发的某个时刻，如果需要的话，产品就可以交付使用了。虽然这一版本的程序可能不会拥有所有功能，但我们知道程序是可以运行的。

采访者：这种开发方式的缺点是什么？

萨奇：如果编程的人超过1个，这种方式就不太好用了。最多只能有3个程序员，而且也不是随便找3个程序员就行了。如果是大一些的项目，就必须采取团队的方式，我们做Jazz项目就是这样的。我并没有反对团队工作方式的

意思，只是对我来说它不是最好的工作方式。

采访者：你仍用这种方式来开发程序吗， 就是一层层地去构建？

萨奇：是的。如果只是我自己一个人开发，我总是这样做的。刚开始先让程序能够运行起来，然后再添加新功能。

采访者：当时你预料到Lotus 1-2-3软件会获得如此巨大的成功吗？

萨奇：米切对Lotus 1-2-3软件信心十足。我的态度是，成功固然好，不成功也无所谓。我仍旧拿我的薪水，对我来说不是什么大的风险。我记得当米切想获得全额投资时，他告诉人们，这个产品将在计算机界引起轰动。说实在的，我不认为他真的那样想，那只是一种炒作而已。但结果Lotus 1-2-3确实引起了轰动。

采访者：当Lotus 1-2-3真正开始成功时，你是处于震惊的状态呢，还是接着又去做其他东西去了？

萨奇：说实话，对我影响不大。在版本1面世后，我在 1983年2月马上开始了版本1A的工作，在其中加入驱动程序，这样程序就可以移植到其他机器上了。我们聘用了更多的程序员做移植程序，并请了几个新的经理接手处理许多日常事务。我们中的几个人在利特尔顿找了一间办公室，在那里开始了Lotus Symphony的开发工作。

采访者：这样做是不是因为你想摆脱在莲花公司办公室里的所有干扰？

萨奇：是的。当你在做像Symphony那样紧张的软件项目时，思想需要高度集中，因此需要一定程度的安静。你必须远离电话和随时进门来的人。但是在Symphony项目做到一半时我就腻了，我已经做够了电子表格软件，而且我也不是很满意Symphony项目的开发方式，它好像越来越庞大、越来越复杂了。我喜欢的是小而简单。所以又回到了莲花公司在坎布里奇的办公室，参与到其他项目中了。然后到了1984年底，我离开了莲花公司，创办了自己的公司。

采访者：你现在在做些什么呢？

萨奇：我所涉及的唯一有商业价值的是字处理器。我用了大约一个月的时间开发出原型。当时我并不认为莲花公司会对它特别感兴趣，但他们显然是感兴趣的。刚开始这个字处理器的功能不多，随后越做越大了。现在我几乎把



所有开发工作都转包给其他程序员了，自己只担任项目的系统架构师。另外，我成立了一个私人基金，做一些环保工作，我还在这里参与了新英格兰地区的环保运动。这些都占用了一定的时间。我现在更多是做一些娱乐类型的计算机工作。我现在甚至会给自己放假了。

采访者：你是精疲力尽了，还是只需要休息一下？

萨奇：我在开发Lotus 1-2-3软件的那10个月，除了吃和睡，一直都在工作。因为时间不够，其他什么事情都顾不上了——我现在还有一些这样的感觉。在大公司担任重要职位有很大压力，你不能松懈。我曾有过在麻省理工学院持续奋战一两个月的时候，但从来不曾持续高强度地工作10个月。在那种压力下连续工作好几个月，在很多方面来说都是一种自我毁灭。

采访者：看起来，这种编程所需要的集中精力和精神高度紧张对很多人来说都不是一件好事。



萨奇：嗯，很多人在气质上不适合。他们会觉得烦躁或紧张。

采访者：你是喜欢编程呢，还是更喜欢做设计或架构？你会每天都坐下来写代码吗？

萨奇：我喜欢做完整的项目，从设计到实现。我喜欢掌控项目中所有的部分。我不是很适合做团队中的一个成员。

采访者：在开发Lotus 1-2-3时，最大的问题或者说最艰难的部分是什么？

萨奇：说实话，从技术角度看，Lotus 1-2-3并不复杂。我不需要重新发明什么，而是利用了一些以前用过的代码和想法。在技术方面唯一让人感兴趣的地方是重新计算的自然顺序以及算法。这也不是我发明的。有一年夏天我们聘请了里克·罗斯（Rick Ross），后来发现重新计算的问题与他所做过的博士论文课题是相似的，与LISP中的垃圾回收有关。他用以往的研究成果对Lotus 1-2-3中重新计算的公式进行了修改。

采访者：你为什么不在大学当数学教授呢？

萨奇：数学太难了。我的数学只能学到我能够直观地解决问题为止。如果太抽象，就做不下去了。但任何东西，只要能在脑海中成型，我都能做出来。

采访者：你在编写计算机程序时就是这样做的吗？

萨奇：也许吧。在你编了15年的程序后，你已经意识不到这个过程是怎样的了。

采访者：你现在编写程序比15年前轻松吗？

萨奇：编程过去15年间似乎并没有什么变化。在具备一定的经验后，你从一个想法开始，一直做到编码结束，都用不着思考中间的每个步骤，整个过程就自动完成了。

采访者：是什么让人成为好的程序员？关键是天赋还是培训？

萨奇：是天赋、气质、动机和努力工作的结合。我知道有很多人期望在很短的时间就成为一名好的程序员，但能成功做到这一点的人并不多。成功来自一遍又一遍地做同样的事情，每一次学习一点点，下一次都做得更好一点。我非常幸运，在职业生涯初期，可以做很多涵盖面很广的、有趣的工作。现在人们很难获得广泛全面的工作经验了，因为工作已经被细分了。人们往往在很早就开始专攻某一方面。



采访者：在你看来，编写不同类型的程序差别很大吗？比如说一个科研项目的编程和编写电子表格软件之间差别很大吗？

萨奇：我发现，几乎所有的编程都是惊人地相似。有一些基本的算法、循环和条件语句，但最终都会归结为同一过程。

采访者：你通常会在程序中添加很多注释吗？

萨奇：多年来我形成了一种注释风格。我会把大部分程序分解成相当小的模块，我会写很多注释，对模块、输入和输出进行描述。这就是我做的所有注释。程序如果小到那种程度，再为所有模块内部的工作机制编写文档就没有意义了。其他程序员只看代码就能看出其中的机制。我的代码就是这样注释的，所以我几乎总能看懂我在很久以前写的代码。

采访者：你在工作中有什么惯常的做法吗？

萨奇：我不是工作狂，我喜欢每天工作一点点。在开发Lotus 1-2-3的时候，我每天都做很多工作。我最好的工作状态是在清晨，这和人们印象中的程序员刚好相反。

采访者：你在家里办公还是在办公室工作？

萨奇：我曾经在家里短暂工作过一段时间，但我几乎可以在任何地方工作。一般来说，我在有噪音或没有噪音的环境下都能工作，但我觉得人的声音是不可思议的干扰，不过我可以在任何其他噪音中工作。

采访者：当你看程序代码时，你觉得什么是具有美感的？

萨奇：如果用C语言来编程，我认为恰当的缩进是非常重要的，变量的命名要便于记忆，这样就知道变量是做什么的了。好的程序是简单而对称的。

采访者：有人在看了你的一个程序后，能认出是你写的吗？

萨奇：有些人能认出来的。我想如果你熟悉某些人的编程风格，通过观察程序中的一些特征就能知道这是谁写的程序了。

采访者：你编程的方式是什么？

萨奇：首先从一个基本的程序框架开始，然后不断添加新功能。此外，我尽量不使用语言或程序中很多花哨的功能。例如，我使用的文字编辑器还是我15年前在麻省理工学院写的那个编辑器的衍生版本。虽然只有几个简单的命令，但我需要的都有了。它现在是用C语言编写的，所以我在每一台新机器上都能用它。我不喜欢使用任何不是我自己写的工具或程序，也不喜欢使用那些我不能全面控制的工具或程序。不然的话，如果某些部分不是我喜欢的方式，我也无法修改它。我喜欢让程序保持简单，这是我编程的一个原则。



有些人很擅长优化每一步指令。他们可以让一小段代码变得极为紧凑。而另一些人只注重算法和实现过程。我介于这两者之间。我不是很擅长非常紧凑地压缩代码。多年前我就发现，如果那样编程的话，每次需要修改代码时，都必须拆开整个程序并重写一遍。但如果稍退一步，只在非常重要的几个点上让代码非常紧凑，那么程序在完成后就容易维护得多了。

采访者：有没有人对你的编程方式产生特别的影响？

萨奇：有很多人。第一个对我产生很大影响的人是埃里克·詹森（Eric Jensen）。他参与了PDP-1上的工作。他比我大四五岁，与他相识是在使用麻省理工学院第一台计算机的时候。他非常有趣，我周围像他那样的人屈指可数——在某件事上达到了非常高的水平，让人觉得不可思议。

例如，埃里克是我见过的在压缩特定的代码方面最棒的人。早期的计算机主机有很多灯和开关。他可以在开关上输入程序。他坐在控制台上，一只

手放在按钮上，把开关的内容输入到内存的下一个位置中。他用一只手翻转所有的开关，就像弹钢琴，每个和弦弹几秒钟。他真是太出色了。我的美学价值和有关做事方式的对错感都是从他那里学来的。

我也通过阅读别人的程序学到了很多。例如，我阅读了IBM的科学子例程包，那是一个庞大的FORTRAN语言程序包，我从中学到了很多编程技巧。我还读了很多书，从中吸取他人的思想。我不是一个特别有创造力的人，我的强项是把好的构想整合起来，使它们成为一个软件包。这并不是说我不能发明新东西，只是说我在那方面不是很出色。

采访者：你认为计算机今后大体上还会继续做现在所做的事情吗？

萨奇：现在创新的速度相当慢。每隔十年只有少数几个真正的新构思。人们会抱怨昔日使用的像纸带之类的东西，但事实上有些老技术是非常好的。我并不确定随着时间的推移，计算机的发展是否会取得长足进步。

采访者：你为什么会这么认为呢？



萨奇：嗯，让我们来看看实际速度吧。比方说PDP-9。它的速度和PC差不多，但有一个速度很快的固定磁头硬盘。它的操作系统非常简单，虽然没有当前操作系统的某些功能，但运行速度非常快。它只需几秒钟就可汇编好一个相当大的程序。即使在IBM 的AT机上，做同样的工作也得等上几分钟。随着技术的改进，我们取得的大多数收益已被通用性的增加和系统的低效率抵消了。而如果用汇编语言编写程序并接管机器的控制权，仍然可以得到非常好的运行效率。其实Lotus 1-2-3软件就是那样工作的。

例如，如果Lotus 1-2-3软件只使用操作系统上提供的工具来完成，它的速度会非常非常慢。总的来说，目前还不是很清楚计算机是否真正取得了进步。在开发了Lotus 1-2-3软件后，我以为有比PC或AT更好的软件开发环境。因此，我们买了些SUN工作站。我发现它们是巨大的倒退。它们的编译速度一点都不比AT的快。我们看到了所有这些新的处理器，但它们的处理能力却在丧失，因为每个人都想要所有的功能，从而总的效率都降低了。

采访者：你认为将来有一天每个人家里都会有一台计算机吗？

萨奇：只要计算机变得更小、更容易得到，就会实现的。已经传闻有很多计算机硬件公司正在研制一种新的芯片，将具有IBM PC机的处理能力，而这种大芯片将取代现有计算机上所有的芯片。所以我认为计算机会进入每一个家庭。

采访者：你对人工智能怎么看？

萨奇：我觉得这个词被用滥了。在编写智能软件时，你会不可避免地遇到难以逾越的障碍。你可以很容易地大量开发程序，但很快会发现，事情突然间就变得极为困难了。

比方说，麻省理工学院想让计算机识别莫尔斯电码。那不就是些折线和小点嘛，但是当由操作员人工发送时，有时发送的点会长些，有时发送的折线会短些。就算折线和点写得不标准，人们也能正确理解含义，但解释那些点和折线就好像是在阅读一样。在阅读文本时，看的不是单个字母，因为字母可能会有裂痕或断开。你看的是单词和句子，那样才会知道整体的意思。这就需要引入庞大的附加信息。就识别莫尔斯电码而言，并不仅仅是找出什么是点、什么是折线的问题，而是要弄清楚整个句子说的是什么、是什么意思。这个领域从来就没有取得过进展，如何去理解含义一直是开发智能程序的主要绊脚石，无论是语音识别、计算机辅助语言翻译还是字符识别。从广义上讲，没有人知道如何编写一个程序来获得经验和学习新东西。



采访者：你是如何看待计算机的？计算机只是一部机器，还是只是一个巨型计算器？

萨奇：客观地说，计算机当然是工具。但计算机也是玩具，很好玩。你可以和一台计算机交互。如果使用得当，还可以做出一些美妙的事情来。

续写传奇人生

当时萨奇已经离开莲花公司，成立了自己的新公司。萨奇受概念性程序“ThinkTank”的启发，开始开发类似的程序，并加入了更多字处理功能。他向创办莲花公司时的合伙人米奇·凯普介绍了这个新软件，这就是后来的Lotus Manuscript软件。此后，萨奇依然想做一个新的电子表格产品，他联系了本·罗森（Compaq主席，曾投资莲花公司）和其他一些人，但这个项目最终未成型。于是萨奇回到莲花公司，担任Lotus 1-2-3/G的产品顾问。

从1990年开始，萨奇迷上了绘画并开始自学。绘画的时候他会拍照来作为参照，也因此对摄影产生了极大的兴趣。他设计了一个图像电子表格软件，并开始四处寻找发行商。柯达公司对此感兴趣，于是萨奇开始在柯达公司担任顾问，并在波士顿招募了团队。一两年后，柯达公司结束了这一项目。萨

奇要回相关技术，并开始开发Windows版本（此前他已在柯达公司开发了Mac版本）。1993年，萨奇创办了一家新公司 Digital Light & Color，向摄影师销售图像编辑软件，但软件销售并不顺利。几经周折，最后萨奇的一位朋友找上门来，于是两个人在业余时间运营该公司至今，只要通过电子邮件来做销售和客服。

萨奇曾提到他自己并非电子表格的用户，虽然有很多人使用Lotus，但他感觉自己与此无关，而图像编辑软件则为他的创造力提供了出口。

萨奇积极参与环保事业。在1985年的时候，他就成立了萨奇基金，资助新英格兰地区、中美洲和南美洲的土地保护活动。他也曾担任 “维护自然环境协会”（Wilderness Society）和“生物保护法基金会”（Conservation Law Foundation）理事会成员，并与“自然保护组织”和“世界野生动物基金会”密切合作。





11

雷·奥奇



雷·奥奇（Ray Ozzie）出生于1955年11月20日，在芝加哥近郊的帕克里奇（ParkRidge）长大。他大学就读于伊利诺伊大学香槟分校，学习计算机科学。在校期间，他开发过PLATO（Programmed Logic for Automatic Teaching Operation，自动化教学操作程序逻辑），那是一个与世界各地近1000个终端相连的计算机辅助教育系统。

1978年大学毕业后，奥奇加入波士顿附近的一家小型计算机公司Data General，在乔纳森·萨奇麾下工作，一起开发一个小型业务系统。离开Data General后，他进入发明VisiCalc的Software Arts公司从事微型计算机和软件工作。在Software Arts工作一年半后，奥奇离职并加入莲花公司，与乔纳森·萨奇和米切·卡波共事。他在那里参与开发Symphony，并且后来成为这个项目的负责人。Symphony完工后，奥奇创办了自己的公司Iris Associates，与莲花公司签订合约开发软件。

目前，雷和妻子朵娜·布斯克（Dawna Bousquet）以及儿子尼尔（Neil）一起住在波士顿郊外的乡村。

奥奇年轻、聪明而又热情。我们约在莲花公司的新大楼碰面，尽管他现在创办了自己的公司，不过仍跟莲花签有合约。创办自己的公司历来是他职业生涯中的目标。他特别强调这要归功于他妻子朵娜，她的耐心和理解使她

能够忍受雷长期枯燥的工作。雷有着深深的眼窝、棕色的头发，前额部分向上卷着，外加整齐的络腮胡，这让他看起来像个欧洲人。他友好而坦率，就自己的编程经历侃侃而谈数个小时，从大学时开发PLATO系统、在莲花公司编写Symphony，一直到创办自己的公司。他不仅就计算机软件的未来发展方向阐述了自己的深刻见解，还给新入行的程序员提供了许多建议。

*　　　*　　　*

采访者：编写好程序有什么技巧？

奥奇：我推崇严谨、一致而清晰的结构。另外，我认为软件应该高度模块化和分层，非常灵活地运用大量文件和目录。如果你必须分别构建不同组件，那么接口自然就会更加凸显出来，要求你规范这些接口。

当许多人共同开发一个程序时，在项目早期确立全局的错误处理、参数传递和子程序命名规范（虽然不见得所有人都会同意），显得非常重要。不过，对于代码该怎么注释、大括号怎么用或者代码该怎么缩进，我觉得你绝不应该对别人指手画脚。在修改别人的模块时，你最好还是按照他的惯例编写代码。这是学会与他人共事的必修课。



交流想法的环境应该是开放的。设计会议上允许非常激烈和紧张的讨论。我发现许多优秀设计师往往固执己见，但只要他们觉得是对的，就会毫不客气地坚持，但他们也知道什么时候应该怎么让步。优秀设计师不会有“非此创造”综合征[①]。

采访者：现在做的这个项目的想法，你是怎么想到的？

奥奇：我还不能公开地评论现在做的项目，不过，我敢说大部分产品都是经过逐次逼近和逐步求精才设计出来的，靠的不是某个人的“灵光一现”。这个项目一开始的设想是打造一款产品，帮助人们在其业务中更有效、更愉快地使用计算机。

人们有一种倾向，认为编写程序只应以赚钱为目的，而不是解决问题。他们不是想着如何创新，而是只顾捡现成的东西，并想方设法抄袭它的思路，心里想着：“哎呀，我可以做个更好的。”人们总是试图构建更好的电子表格，

① “Not Invented Here” Syndrome，即NIH综合征，指的是社会、公司和组织中的一种文化现象，人们不愿意使用、购买或者接受某种产品、研究成果或者知识，不是出于技术或者法律等因素，而只是因为它源自其他地方。这个词通常带有贬义。

大量精力和资金就这样白白浪费了。但这个世界用不着57种电子表格。我觉得我们现在见到的是业界最后一拨要把电子表格做大的人了。一旦百万富翁的梦想破灭了，人们就会更加重视垂直市场，针对特定应用开发软件，而不是好高骛远，提供人们不需要的产品。

我们的产品不是盲目跟风货，我相信它会令许多人受益匪浅。不过，只要它能让用户感到快乐，并且更加富有成效，我们也就完成了使命。

采访者：提出软件产品的想法时，你确信自己能够实现并按时交付吗？

奥奇：是的，我非常肯定自己可以做到。我已经有足够的经验，非常清楚自己能做什么。非程序员出身的管理人员负责的复杂编程项目往往注定要出差错，因为他们既不理解项目组件之错综复杂，也不了解程序员的个性脾气。软件项目经理必须熟悉手下的员工。我会尽力了解与自己共事的每个人的家庭状况、生活方式以及工作习惯。我知道，要是每天朝九晚五地工作，项目肯定完不成。另外，我也非常清楚，不能逼着大家在整个项目期间没日没夜地工作。但我相信，遇到紧要关头时，如有必要，我可以依靠他们夜以继日地工作。我还得知道什么时候该让大家放松一下。



另外，我还喜欢跟阅历丰富并有过相关经历的人共事。我会请程序员预估他完成某项任务需要多长时间，并根据他预估的时间做出规划，然后按我以往跟他共事的经验进行调整。许多初级程序员预估的时间可能偏差很大。之后，随着经验的积累，他们会根据自身相对稳定的工作习惯和当前积极性，把握自己的“修正系数”。

采访者：听起来，管好程序员似乎是按期完成项目和打造高品质产品的关键所在。你是怎么做的？

奥奇：许多经理发现程序员很难相处。我很少有这类问题。管理人员试图颁发条令或过度管制时，往往很容易滋生问题。程序员很有创造力，善于自我指导和自我激励。你必须预先认识到这一点，不到万不得已，千万不要介入。如果你特意让团队日子好过，他们也能意识到的话，在关键时候他们自然会为你着想。

整个环境必须有利于编程。不同的人对工作环境要求不同。在我们看来，好的工作环境要有漂亮的办公室，要尽可能提供最好的设备，每个房间一台立体声，冰箱里总是装满食物。在Iris公司，有些团队成员喜欢在家里工作，他们在家用的全套机器配置跟公司的完全一样。

程序员不必担心自己做的是否跟他们的同侪（或管理同行）一样多，因为这些不安全感可能会对积极性造成负面影响。他们应该感觉得到自己是团队的一部分。这就是为什么团队规模应该尽量小，并且尽量不要分等级的原因。

采访者：你刚才提到自己偏爱阅历丰富的程序员。具体怎么理解？

奥奇：要有丰富的阅历，你必须做过各种不同的任务。大学毕业后，最理想的选择是先在某个领域认真做上一年左右，然后转到计算机学科另一个截然不同的领域。例如，你可能会先从操作系统入手，然后转到网络、图形、编译器或数据库，以便熟悉各个不同的领域。从长远来看，做一个全面的程序员，你会更吃香，身价更高。

许多人刚从学校毕业，知识面和经验积累都有所欠缺，而这些都是独立完成重要的产品概要设计和开发不可或缺的。阅历丰富的程序员是个多面手，善于协调自己的才能和生活方式，有能力进行抽象思考，能够与他人和睦共事，既自我激励又干劲十足。



采访者：说说你是怎么开始编程的。

奥奇：1969年我刚上高中一年级。有个数学老师上课时带了一台可编程计算器，展示给我们看（好像是一台Olivetti-Underwood Programma 101），想着我们会为这个精巧的小玩意儿兴奋不已。他跟我们说，如果想玩这个计算器可以直接去找他。我们中有几个人确实很感兴趣，我们花了大量时间来研究这个计算器的功能。

我们留意到学校老师有他们自己的小玩具。数学办公室有一台通用电气400型电传打字机，跟学区里的一个分时系统相连。好几个数学老师都在摆弄它，试图了解它是怎么工作的。当然，我们也想摆弄那台电传打字机，于是设法获取了密码，并用BASIC和FORTRAN写了些好玩的程序。我玩得很开心，不过随着其他活动占用了更多时间，我最终对它失去了兴趣。

采访者：其他什么样的活动？

奥奇：上高中时，我对技术工作和电子学很感兴趣。我选修了学校开的所有电子学课程，暑期还去当过电子技工。进入伊利诺伊大学香槟分校时，我以为自己会念电气工程。我大一大二学的就是电气工程。

开始选修更多电气工程的课程之后，我意识到这跟自己原来想的不是一

回事。我喜欢电子设备的修修补补，但对那些数学理论根本提不起兴趣。当我正考虑换个新专业的时候，有个念计算机科学的朋友说服我选了一门编程课程。我非常喜欢。布置给我们的计算机问题既有趣又简单。尽管我并没有理解所有概念，但我喜欢编程跟喜欢电子学有着同样的原因。我可以用程序代替器件搭建小玩意。太棒了。

采访者：你就这样迷上了编程？

奥奇：那些编程课还不赖，但真正让我迷上编程的是学校里一个叫PLATO的计算机系统。发现之后，我就禁不住想用这个系统。我发现有一种方法可以获得访问权限，即选修一门使用这个系统的计算机科学课程。于是我选了一门课，开始折腾这个系统。

这是我编程经历的真正开始。最终，我能以系统程序员的身份访问这台机器，就这样培养出了我的编程习惯。每到晚上10点，我就可以使用这台机器，一小拨像我这样的人会通宵工作，直到早上6点为止。我们有几年都是这么过来的。真的很有意思。

从我能用上那台计算机的那一刻开始，我对功课就再也提不起兴趣。实际上，我甚至觉得自己之所以还待在学校，唯一的原因是我必须注册入学才能保持我的习惯。要是被踢出学校，我就玩不了那台计算机了。我不是个好学生，但还是设法坚持了下来。我大学念了5年半才毕业。

采访者：毕业后你是怎么从学生转变成系统程序员的？

奥奇：使用PLATO期间，我上了政治上的第一课。经过9个月争取以学生身份获得访问权限的努力之后，我开始意识到光是擅长或喜欢编程还不够。要想有所成就，你就必须会玩政治把戏。使用PLATO系统有一套非常严格的权限等级制度，因为想用这个系统的人太多了。

采访者：你认为之所以会这样，是因为很多人想使用这个系统，还是因为PLATO是学校不愿意让人们接触的那类神秘物件？

奥奇：也许两者兼有吧。伊利诺伊大学的PLATO是个计算机辅助教育系统，运行在控制数据公司（Control Data）制造的一台超大型机上，世界各地有近1000个终端与之相连。这个系统由规模很小、关系紧密的一小组人设计和实现。他们不喜欢别人使用他们的发明，在涉及是否准许人们进行系统编程的时候，这点表现得尤为明显。



最初，只有那些设计系统的教职人员，还有几个在这个圈子里混迹多年，且已经成为朋友的高中生和大学生，只有他们才获准进行系统编程。不属于这个圈子的人想在这个系统上编程，必须使用一种叫Tutor的中间层解释语言。我决意要做系统编程，于是我想方设法，排除障碍，进了这个圈子。

采访者：作为系统程序员，你在PLATO上主要做什么？

奥奇：在我选的课上使用PLATO系统时，我听说PLATO硬件组正着手开发一个基于Z80的可编程终端。PLATO工作人员找我开发这个可编程终端的固件。

当时我一直在核工程系做兼职电子技术员，给那里的研究生定制小配件。

后来，到了大四，我和另外两人开了一家小公司，Urbana Software Enterprises，专为一家物业管理公司开发软件。他们打算构建一个微机系统，用在自家的垂直应用中，为此他们委托一个硬件小组开发基于Z80的机器，并委托我们编写语言编译器、解释器和操作系统。



采访者：你念大学用了5年多时间，有什么原因？

奥奇：我每个学期注册的课程数不算多，然后其中又有两三门不上。另外我中途退学了一阵子。我坚持念完大学，是因为害怕自己没毕业的话就会找不到工作。在整个求职过程中，我非常紧张，缺少自信。幸运的是，我找工作那年，1977年，是个好年头，大公司都在积极招人。我面试了12家公司，结果都还不错，主要是因为我的实践经验比较丰富。

采访者：你怎样决定到哪家公司工作？

奥奇：我的大部分朋友都去了西海岸，或者加入了马萨诸塞州梅纳德[①]的数字设备公司（DEC）。我被DEC拒了，真糟糕，因为我本来希望能和自己的朋友一起工作。我觉得DEC是工作的最佳选择，因为可能有机会做VAX软件开发（当时VAX还未正式发布）。

我跟Data General签了合同。接受这份工作之后，我收到DEC的一封来信，说他们想找我再谈一谈。但是我已经答应乔纳森·萨奇到Data General工作。我们准备从零开始设计一个小型业务系统的操作系统和体系结构。核心小组只有三个人。

采访者：再说说你设计的系统。

① 1957年至1992年数字设备公司总部所在地。

奥奇：我们开发的是一个低端小型业务系统。这个系统意在替换现有产品线，屏幕I/O性能可与Wang文字处理机（王安电脑公司推出的产品）匹敌。

我们想出了一个基于Nova微型工作站和Eclipse文件系统服务器的小型系统。每个工作站都自带处理器，有64KB内存（但没有大容量存储器），还有键盘和监视器。它通过我们开发的局域网络连接到其他工作站、文件服务器和打印服务器。我负责开发工作站的固件和操作系统，萨奇和斯科特·诺林（Scott Norin）则负责开发文件服务器。这个系统历时两年开发才算真正跑起来，就在那时，萨奇离开公司，跟约翰·亨德森一起创办了Concentric Data Systems公司。之后，斯科特和我又待了一年，对系统进行最后的打磨。整个部门非常忙碌，这个产品也差不多完成了。

采访者：那么之后你又做了什么？

奥奇：我在Data General开始变得很沮丧。在大公司里想干点事情真让令人痛苦。

因此，大概就在萨奇离职时，我开始跟猎头接洽。我告诉她自己想去一家涉足微型计算机的小公司。在那之前，我就一直在阅读相关杂志，还玩过CP/M，我觉得跟微型计算机打交道似乎很有意思。但是没有一次面试合我心意。大多数公司都从事大型计算机业务，不涉及微型计算机。大约一年后，我接到猎头的电话，她说找到一家小公司，我可能会有兴趣。这家公司就是Software Arts。我到那里接受面试，并且立刻喜欢上了这家公司。于是我加入Software Arts，成为第29号员工。

回头看来，我离开Data General实在是明智之举，就在我离职大约一个月后，那个项目便因为政治原因而被撤销。Data General试图在公司范围内统一采用一个操作系统，而我们的项目与之冲突。它已经完成，而且就快正式推出了。我怎么也想不到会发生这种事情。

采访者：你在Software Arts主要做什么？

奥奇：一开始，我在Radio Shack的TRS-80 Model 3上开发语言解释器，为TK!Solver的实现奠定了基础。这部分工作我只做了几个星期，后来他们为了开发VisiCalc购进一台新机器IBM PC。我想不到安保措施会如此严密。房间全都上了锁，那些手册也不能带离房间。这显然是台原型机。试用几天之后，我发现自己也想拥有一台。



采访者：你认为最终人人都会想要一台个人电脑吗？

奥奇：当然。好吧，我觉得至少程序员会想要。当时我对市场没什么概念，基本上眼里只有编程。如果IBM推出微型计算机，那就意味着价格会下降，计算机会变得更便宜。这也意味着像我这样的程序员不用再依赖大型机。

我对IBM PC异常兴奋，因为我觉得这是第一台足够强大的微型计算机，可以在上面编译自己写的程序。我可以在同一台机器上编辑和编译程序，链接，并执行测试周期。当我们第一次拿到硬盘时，那是美梦成真的一刻。我只需坐在自己的办公室里，就可以随心所欲地工作。这台计算机附带的文档非常糟糕，几乎找不到任何技术资料，不过我实在太高兴了，以致根本没放在心上。虽然微型计算机的速度比较慢，但在它上面工作起来更自在。

我对个人电脑满怀热情，因为酷爱编程的年轻人用不着再像我们那样处心积虑搞政治把戏，也能用上计算机了。我真希望家长和学校认识到个人电脑是多么奇妙的资源。我真心希望计算机中心能为那些想玩电脑但又买不起的孩子大开方便之门。



采访者：你在Software Arts工作似乎如鱼得水。你想过要离开吗？

奥奇：我在Software Arts工作了几个月，有一天，萨奇和我一起吃午饭，他刚刚离开Concentric Data Systems公司。他告诉我他刚认识一个非常聪明的家伙，名叫米切·卡波，那家伙创办了一家名叫微财务系统（Micro Finance Systems）的小公司。萨奇打算跟他一起干。当我问及他们会做什么，他实在不方便透露，不过从他的话里听出来应该是VisiCalc仿制品。他想知道我有没有兴趣到那里工作。

我不知道说什么才好。我刚到Software Arts工作没几个月，很高兴能开发真正的VisiCalc，我们正准备展示Radio Shack机器上运行的版本。对我来说，这是一大步，它意味着VisiCalc将无处不在。要是你能参与原装正品的开发，何必要去折腾仿制品呢？我决定留下来。与此同时，乔纳森和米切继续做他们的事，他们打造了Lotus 1-2-3。

采访者：你下次见到乔纳森是什么时候？

奥奇：下一次我见到他，另外还第一次见到米切，是在COMDEX上宣布1-2-3的时候。当时我在Software Arts工作了一年半，TK!Solver已完工并发布。

就在下飞机的那一刻，我意识到米切和乔纳森（现在的莲花公司）干得

不错。1-2-3是那次展会的热门话题，不过当时我还没想到它会打败VisiCalc。

我跟乔纳森促膝长谈，他再次敦促我加入他们的队伍。当时1-2-3虽已正式推出，仍有一大堆功能有待实现，他和米切希望在1-2-3的第二个版本里加上这些功能。我答应会认真考虑一下。

采访者：你准备离开Software Arts？

奥奇：也许我早就想离开Software Arts了，只是自己没意识到而已。因为COMDEX召开前6个月，我就试图跟来自DEC的三个家伙一起开一家公司。其中一人已经跟国家半导体公司的代表接洽过。据称他们正打算投资一个项目，基于即将推出的16032 CPU，打造一款廉价、低端的类VAX系统。我们准备仿效VAX/VMS为这台机器开发操作系统。

最后这没能实现，因为国家半导体公司下不了决心，无法提供项目所需的几百万美元启动资金。我们并未就此放弃。我们接触了几个风险投资家。这段经历发人深省，因为他们问了许多刁钻的问题，而那些问题我们从未深入思考过，尤其是市场营销方面的。最终我们解散了那家公司。



采访者：你当时认真考虑过乔纳森的提议吗？

奥奇：那时，我妻子朵娜正在待产。尼尔出生后，我们重新考虑这个问题。我还是迫切希望自己开家公司，甚至在努力写一款新软件产品的初步功能规格说明书。在莲花公司同米切讨论此事后，我们商定了一个条件，在此前提下，我会到莲花公司开发后来的Symphony。待这个项目完成后，我们再重新审视我的功能规格说明书和创业夙愿。

采访者：在莲花公司工作是什么样的？

奥奇：开始上班没多久，我发现从住处到坎布里奇（马萨诸塞州）一个小时的路程太费时间，这一个小时本来可以投入到工作中。跟米切商量之后，乔纳森和我在利特尔顿（马萨诸塞州）开了一间小办公室，参与项目开发的三个人，乔纳森、我和巴里·斯宾塞（Barry Spencer），到办公室的距离相当。

采访者：于是你准备参与Symphony开发。当你加入莲花公司时，Symphony的所有构想都已经确定下来了吗？

奥奇：全部加起来只有五六页概要综述，还有一长串数不清的功能希望在产品里实现。

我开始开发字处理器这个主要组件。巴里·斯宾塞也刚参与这个项目，

致力于另一个重要组件即通信系统的开发。几个月后，乔纳森似乎厌倦了这个项目。他的心思没有真正用在他负责的工作上，可能是因为他一直全力扑在同一块代码上太长时间了。于是他不再管这个项目，我成了项目主管。我们找到一个非常出色的程序员，马特·斯特恩（Matt Stern），这样团队又恢复到三人。马特最初负责数据库组件开发。我们三人又发疯似地工作了9个月。我每两周去一趟莲花的总部，当面向米切汇报项目最新进展。他会检查并评估我们的工作，我们会重新调整时间表，然后回去再接再厉忙上两周。

采访者：这种方式工作起来有成效吗？

奥奇：当然。莲花之所以会成为我工作过的最佳公司，米切就是原因之一。他不会对权力下放感到不安。他对功能细节、用户界面以及功能的实现顺序很感兴趣。不过编程仍在我们掌控之中，因为他对算法并不感兴趣。他不会每5分钟就介入一次，没有充分理由也不会变更设计。



采访者：在你看来，复杂如Symphony的程序为什么这么容易就整合到一起？

奥奇：其中一个原因是公司许多人都在倾尽全力开发Symphony。每个人都为这款产品感到非常骄傲，致力于按时精准地完成产品。不管承认与否，我们许多人也担心来自Ovation和Framework的潜在竞争。

我们按期完成的另一个原因是沟通十分顺畅。在封闭环境中，规模小的开发团队沟通非常有效。后来，跟莲花总部的沟通开始逐渐成为瓶颈，我们的应对措施是关闭分公司，搬回总部。

最后一个原因是开发团队规模小。我认为，可能的话，产品的设计和实现不应超过5个人。当然，如果你正在实施某个大型系统，比如美国国税局税务审计系统，用规模这么小的开发团队，显然不切实际。不过，小团队往往占有很多优势。

采访者：规模小的团队比大的团队更有效率，何以见得？

奥奇：我认为团队成员一旦超过5人，成员之间的沟通就会开始变差，而这可能导致产品一致性方面出现问题。

缺陷往往是子系统之间接口糟糕的表征，这通常要归咎于设计人员在设计各子系统时沟通不良。发现缺陷时，人们倾向于在子系统内部加以解决，而不是从整体上处理这个程序。除非这几个人都清楚各个子系统是怎么回事，这些子系统是怎么融为一体的，否则最终产品很可能缺陷重重。

采访者：健壮的程序还有其他重要指标吗？

奥奇：在开发过程中，产品必须具有非常清晰、一致的架构模型，这点很重要。你可以在开发过程中改变架构，但不可以到结束时才用修正缺陷的方式来修补糟糕的架构。你不会只碰到几个小缺陷，而会不得不疲于掩盖软件内部的惊人裂隙。

1-2-3设计精良，架构清晰一致。Symphony是个庞大的软件层，构建在1-2-3这款优雅的产品之上，1-2-3的规模非常大，不同组件之间存在大量交互。它从未出现过重大缺陷，因为所有开发人员都理解子系统之间的交互和1-2-3的基本框架。

采访者：Symphony完工后，你接着做了什么？

奥奇：米切没有食言。我再次拿出自己加入莲花公司之前写的规格说明，我们俩一起讨论了一番。他建议我再花几个月时间整理那些规格，以确保我能描述产品是如何运作的，有什么好处，人们如何使用它，并确定如何进行市场推广。巴里·斯宾塞接手了Symphony的开发工作，使我得以将全部精力投入到新产品中。



在我完成设计规格后，副总裁粗略审核了一遍。我必须回去修订设计，他们也没承诺一定会感兴趣。这个产品必须经得起详细的审查。

最后，公司同意投资我的项目，并以一家独立公司Iris Associates的方式运作。我同之前的伙伴蒂姆·哈尔沃森（Tim Halvorsen）和伦·卡维尔（Len Kawell）立即加入Iris，他们俩都来自DEC。那之后，我们又雇了两个人。

采访者：你认为5到10年后市场上会出现什么样的程序？

奥奇：我都等不及想看看它会是什么样的。我过去经常自以为可以想见它最终的样子，但现在毫无头绪。像其他很多人一样，我曾错误地相信会出现家用电脑市场之类的。

我过去常想，个人时间管理程序应该会非常便利。结果现在却得在公文包里装一台计算机随身携带，打开并启动计算机，就只为了查看我的会面安排，远不及我那口袋大小的通讯记事簿来得方便。我想要一本通讯记事簿，让我的秘书和我可以同时写入，这非常有用，因为我们俩都能便捷地安排我的时间。我希望哪个聪明的家伙能发明这种记事簿。

采访者：对那些想投身软件开发的人，你有什么特别的建议？

奥奇：那些了解垂直市场用户的人机会很多。像1-2-3这种销量大的水平应用屈指可数。眼下，你也许一只手就能数过来：字处理器、电子表格和数据库管理系统。如果从长远来看个人计算机将获得成功，那也是因为程序经过细心剪裁，能满足特定用户的需求。

那些在他们父母的企业中工作过的计算机科学毕业生拥有编写垂直软件包的绝佳机会。他们懂计算机，很清楚它们可能会对某项具体业务产生怎样的影响。我觉得好高骛远的人应该少些，更多的人应该找准一块细分市场。

采访者：开发产品时，你觉得考虑最终用户的需求有多重要？

奥奇：我工作是因为我喜欢玩电脑，但是，我设计产品是因为我觉得自己能为消费者提供有用的东西。在整个产品开发周期中，极为重要的一点就是始终牢记最终目标。

采访者：你们如何向最终用户提供他想要的东西？



奥奇：首先，我们会尽量设想出哪些用户会使用该产品。然后，我们设法粗略估计使用每个功能的用户占比，这样才能有根据地猜测哪些特性的使用频度最高。我们尽量把大部分时间用于完善使用比例最高的特性。如果某个不起眼的功能只有小部分用户使用，我们就不会为这个功能的设计大费周章。只有等产品推出之后，我们才能确认对用户的设想是否正确。

采访者：你有没有收到过用户的反馈？

奥奇：有啊。这方面的事例很多。从每天使用你的产品并发现某个边角功能存在缺陷的长期用户那里收到报告很有意思。想到其他地方生活方式完全不同的人也在使用你的软件，那有多棒；再想到他们一定是觉得产品还不错，所以才会愿意花时间写信发牢骚，这真是令人心满意足。

不过，最奇妙的经历是刚创办Iris公司那阵，我接到一位外科医生打来的电话，他在心脏直视手术过程中使用Symphony分析实时数据。这让我清醒地认识到，有人躺在手术台上，可能全靠我的程序正确运行。这提醒我要对最终用户切实负起责任。

采访者：你觉得编程好玩吗？对你来说，写程序让你难受还是愉快？

奥奇：不愉快的话，我就不会干这行了。在紧张的开发过程中见不到朵娜和尼尔，这很让人难受；我恨不得一天多出几个小时来。没有朵娜的倾力支持，

我绝对干不了这行，她的作用无可替代。

采访者：编程的魅力何在？在我看来，许多程序员天生就像修补匠或工匠，他们喜欢搭建东西，后来又对编程产生兴趣。为什么会这样呢？

奥奇：编程是热衷修补者的最终归宿。修修补补离不开工具。电气工程师可以把各种组件放在一起搭建东西，但他们受制于物理设备是否容易获取。而有了计算机，只有你想不到，没有做不到的。你可以设计自己的工具，或边做边打造部件。不喜欢某样东西的话，你可以直接修改或重写它。拥有计算机这个资源，就等于有了完全开放的工具箱。唯一的限制因素是计算机执行任务所用的时间和你编写程序所需的时间。你拥有不可思议的灵活度。

采访者：你怎么防范职业倦怠？

奥奇：时间不要排得太紧，开发过程中要合理安排休息次数。如果打算集中一段时间进行编码，我会排好时间，在开始下一轮重负荷的编码工作之前，留足6个月的时间做些概要设计之类的工作。我会让自己休息一阵。等到重新投入高强度的编程时，我已经巴不得早点开始了。



采访者：你对现在的年轻程序员有什么建议？

奥奇：如果你是硬被拉入编程这一行的，我建议你保持乐观的心态，尽量多编程，同时尽可能参加各种不同的项目。尽可能多把时间花在计算机上，还要学会准确判断自己的职业倦怠状况（burnout level）。要是别人觉得你很古怪，不用放在心上。

续写传奇人生

1994年，奥奇任CEO的Iris公司以9400万美元的价格被莲花公司收购。1995年，IBM又以35亿美元的价格买下莲花。奥奇在IBM Notes部门待了两年的时间，于1997年离开。虽然他不喜欢IBM，但认为这段时间是值得的。他曾这样表示："IBM接手Notes的时候，Notes有200万用户；当我离开的时候，Notes有1.5亿用户。这在莲花公司绝不可能发生。"

离开IBM后，奥奇创办了Groove网络公司，开发网络协作软件。2001年，Groove 1.0发布，基于P2P技术的Groove不仅有强大的跨群组网络协作功能，还有强大的加密设计。但恰逢后泡沫经济时代，奥奇意识到只靠自身

无法坚持下去，最好的选择就是将Groove卖与微软，要知道此前微软已经在Groove上投资了5000万美元。

2005年3月，Groove被微软并购，奥奇加入微软。当时微软的关注点是大型软件，即重新改造的Office和Windows Vista，但奥奇认为这些并不是微软应该努力的方向。他决定将他的想法写下来，于是有了后来的《互联网服务分裂备忘录》。这份备忘录被盖茨转发给了100位高管和经理。奥奇备忘录是一个明确的行动呼吁，同时也是对微软当前状态的微妙批评。奥奇写道，微软必须要像互联网公司一样思考和运营。这份备忘录不仅仅开启了Windows Live，使Office走向云，而且开启了Windows Azure，将公司产品从软件转向服务。奥奇还创办了微软未来社会化体验实验室FUSE Labs，该项目的目的是更好地理解社交计算。

2006年6月，奥奇接替盖茨担任首席软件架构师。2010年10月18日，奥奇宣布将暂时转任公司娱乐暨设备部门，并计划一段时间后退休。与此同时，奥奇在四年的间断之后又开始写博客，并于25日发表了另一份备忘录。他表示，微软在过去5年中没有充分实现他提出的构想，错过了机会；一些竞争者在软硬件结合、社交网络以及移动服务方面已超越微软。他呼吁微软必须"为后PC时代勾勒出一幅务实的前景"。在奥奇计划退休时，微软CEO史蒂夫·鲍尔默表示，微软将不会寻找奥奇的替代者，也就是说，微软今后将不再任命新的首席软件架构师。



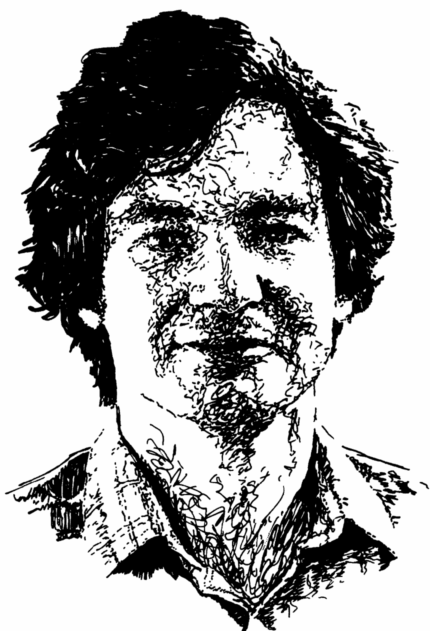
This is the main loop for the edit command in our newest version of T/Maker which also supports Japanese and Chinese (thus the reference to "kanji").

I would say this code is pretty typical in terms of how it looks.

```
edit ()

 { inedit = 1;
   oops[0] = NUL;
   tcsd2 = tcs/2;
   move (tl1,tc1,1);
   pmenu (400);
   if (noimage)  gdraw (0,1,1,1,tl1,tl2);
   else          gdraw (0,1,0,0,tl2-3,tl2);
   error = maccol = maccnt = noprint = mmode = 0;
   inmac = chchin = rinfo = 0;
   chr = 390;
   for (;;) {
         if (rlines) {
            if (ccol >= tc1) ruleone (ccol,0,0);
            else             ruleone (ccol,1,0);
            pblip();
         }
         if (chr == 384) break;
         if (chr < 32 || chr == 127)   ederror(65); else
         if (chr >= 527 && chr <= 529) macro ();    else echar ();
         if (theline[ccol] > SPACE && ccol)
            if (callutl(kanji3,0,theline,&theline[ccol-1]))
               if (ccol < tc2) fs(0); else bs();
         if (rlines) {
            if (ccol >= tc1) ruleone (ccol,0,1);
            else             ruleone (ccol,1,1);
            pblip();
         }
         if (rinfo++ >= 50) { rinfo = 0; dinfo (1); }
         else           if (lastinfo != cline) dinfo(0);
         if (error) break;
         pblip();
         chr = gfunc ();
    }
    inedit = 0;
    clrbl();
    mmode = 0;
    dinfo (1);
  }
```

用C写的样例代码，可以说明彼得·罗伊森的编码风格。



12

彼得·罗伊森



作为一个加利福尼亚本地人，彼得·罗伊森（Peter Roizen）就在帕洛阿尔托长大。他在加州大学伯克利分校就读，并于1967年获得数学专业的学士学位。他毕业后的第一份工作就是程序员，虽然在此之前他几乎没碰过编程。罗伊森在离开伯克利后在蒙特利尔和多伦多待了两年，然后去了欧洲，在世界卫生组织干了七年程序员。他后来回到美国，在华盛顿特区为世界银行工作。1980年，他创办了自己的公司，推广和销售他的电子表格程序，即T/Maker。这是当他在世界银行任职时用业余时间开发出来的。1985年，罗伊森把他的小公司从华盛顿市搬到了旧金山湾区。他现年39岁，已婚，有一个5岁的儿子。他住在加利福尼亚的洛斯加托斯。

我和彼得约好了在他的公司T/Maker的办公室见面，尽管他在那里的时间很少，经常一周只有一天。实际上，他在办公室的环境里显得格格不入，对充斥着办公室的电话铃声、说话声和忙于销售软件业务的人显得很不适应。看起来他像是刚从冬眠里醒过来和我会面似的。罗伊森更喜欢在他家中的办公室里独处，一个人开发软件。

他已经年近四十，看起来很安详。头发蓬乱，呈深褐色，脸很有棱角，经常会露出温暖的笑容。他很看重自己的独立性，很高兴自己可以不依赖于

公司和薪水，可以靠自己的能力做自己喜欢做的事情——编程——来谋生。对于怎么做一个程序员他也并不拿腔作调。他没有大肆吹嘘他的职业，也没有想要开发出一个划时代的软件来发财致富。恰恰相反，他认为自己只是非常幸运，发现了他在世上的合适位置，并靠此安身立命。

* * *

采访者：你为什么喜欢编程？

罗伊森：噢，我一直很喜欢像国际象棋和双陆棋这样的游戏，因为这些游戏需要战略考虑才能赢，而且在游戏结束时我总能知道我是赢了还是输了。编程很像游戏。如果我只是想不花太多力气写一个程序来解决一个问题，那就不会有很多决定要做，我通常可以比较直接地解决问题。但是，总会有不同的方向可以选择。编程很有趣，就因为要作出最好的设计决定极富挑战性。编程非常好的一方面就是，当你干得很好时，结果会非常清楚。当我挂墙纸时，如果我非常注意，我会干得不错；但如果我马虎，结果就会很差劲，边角会翘起来。如果我编程的活干得不错，程序运行一秒钟就结束了，任务完成。如果我活干得很差劲，程序会跑上十秒钟，全部东西死翘翘的情况每二十次里就会发生一次。我喜欢知道自己是否完成了目标。



采访者：所以，可以尽你所能，这就是编程让你感到满足的地方？

罗伊森：当然。不过，我不想言过其实。这个世界非常大：再过一百年，人们甚至都不会知道这个程序是什么。我并不想要影响世界的进程。我只是想每天都做一些有价值的事情。这样，当我干完时，我不会觉得虚度光阴。

采访者：你是不是觉得完全放不下编程？

罗伊森：不是这样。任何时候我都可以放下编程。我并不觉得我隔一段时间就得碰一下键盘，或者非要一直呆在计算机旁边。有时候，当我不想编程时，我会做一些其他事情。我过去常画漫画。我不太会画画，所以要画出我想要的效果有点难，但画画还是很有趣。我也喜欢滑雪，虽然不是什么高手。

现在，照顾我五岁的儿子、帮做一些家务占掉了很多的时间。但我还是能从编程中获得很大的满足感。跟其他领域比，这也没什么特别好的。只是它恰好适合我，让我感到快乐。所以我就一直坚持下来了。

采访者：你可不可以描述一下你是怎么编程的？

罗伊森：我是个非常“自底向上”的程序员。我不为整体战略伤脑筋，因为除非完成了所有细节，否则我从来看不清全貌。如果你想要把工作交给其他人做，“自顶向下”的编程自然很好——很显然，你需要了解各个部件是如何在一起工作的。对我而言，编程是一个反复迭代的过程。我总是一次做一个程序的一小部分。如果我在一个地方改动一下代码，就很可能需要在其他地方也动一下。我会这样上下调整，直至方案合适为止。

我也非常实际。我见过很多不实际的聪明人，他们在追逐一些完全无用的学术目标时迷失了方向。实际对编程很重要。你必须能够猜出完成一个项目的准确时间，具有推断可以做哪些、不可以做哪些的能力，然后据此去做，抵制中途停下来干其他事情的诱惑。当你想做好每一件工作时，实际一些尤为重要，毕竟你可以分配在每件工作上的时间很有限。如果你想把一个项目做到尽善尽美，往往轻易就会多花上两倍的时间。可东西如果没完成的话，就不会卖得好。没有可销售的产品，你就很难维持生计。



采访者：你在编程行当已经待了18年还多。你最初是怎么进来的？

罗伊森：1967年拿了数学学位从伯克利毕业时，我去找工作，直接就接受了第一份给我的工作，去做程序员。那时我对于编程几乎完全没有经验。那时甚至还没有计算机科学系。我上过一门FORTRAN课，也曾经给一门编程课做过助教。因为我得给学生提供帮助，批改他们的卷子，我必须得比他们超前一点。这就是我在这一领域的全部经验了。

采访者：你的意思是你在做这份编程工作之前就没写过程序？

罗伊森：那时这个领域还很新，雇主会招聘在编程方面有潜质但缺乏经验的人。像很多人的第一份工作一样，我根本不在意它具体是什么。我使用FORTRAN和COBOL，边查手册边写程序。

采访者：然后你就一直做编程了？

罗伊森：在第一份工作之后，我在欧洲为世界卫生组织工作了7年，然后在华盛顿的世界银行工作了7年。这两家都是非常大的机构。随后我在1980年创办了自己的公司。

采访者：你在开发程序时遵循某些原则吗？

罗伊森：我相信使用自己写出来的东西非常重要。并不是说我是自己的程序的最好用户，而是因为经常使用的话我就可以意识到程序中的缺点，可以在发现什么地方做得很笨拙的时候就把它改掉。我记得有一次在世界银行的时候，我向别人演示一个程序。程序是别人写的，而且之前我没用过。第一行显示在屏幕上时，提示说可以输入HELP来得到帮助。我想，这是一个很好的开始，就输入了HELP进去。结果我得到了这样的错误信息：FATAL ERROR--SEE COMPUTER ACTIVITIES PROGRAMMING DEPARTMENT（致命错误——请联系计算机活动编程部）。这说明设计师根本没好好用过他自己的程序，甚至没有照着他写在屏幕上的第一条指示操作过。这种做事方式非常危险。

采访者：程序员怎么会从来没试过他自己的程序呢？



罗伊森：在以前，很多公司会把分析师和程序员区分开。分析师使用流程图做出总体规划，程序员填补剩下的细节。如果你想解决问题，却碰到一拨人只管画方框、另一拨人只管填补方框里的内容，你最终将不可避免只能拿到一堆不能解决问题的垃圾。在我做程序员时，我从分析师那儿拿到的方案有一半是不可行的。他们了解全局，但没有足够的具体信息来做好他们的工作。没有人聪明到看一下整体图景就知道程序需要操作的具体细节。所以我赞成一个人把整个程序从头写到尾。

采访者：你认为公司里的程序员必须要妥协吗？

罗伊森：不一定。有人付你薪水时，你总要在某种程度上做他们希望你做的事情。我们得面对这一点：金钱能买到影响力。这是这片土地上的规则。但我不会把这叫做妥协。事实上，有些公司把这称作成熟，认为这是积极调整。当我为这些大机构工作时，对于某些分派到的工作，我做得很开心，对结果也很满意，没有什么需要妥协的地方。

采访者：你是怎么写程序的？

罗伊森：因为我一般所有的东西都自己写，所以就没有时间去实现太过复杂的功能。我必须在合理、有限的时间里完成程序的开发。只写规格书的人就

不怎么关心时间了。在想做的事情和当天可以完成的事情之间有着很大的区别。我们公司没有时间、也没有金钱来做我们所有想做的东西，在某种程度上，这是一种阻碍。但是，从另一方面，这也有好处。我们的程序不得不保持得相对比较苗条、比较有用，因为我们决不会添加一堆肯定没用的功能。

采访者：*你会在代码里写很多注释吗？*

罗伊森：几乎完全不写。如果我因为什么原因需要给别人一些源代码，我会很快回去把注释加上。我能和自己的作品直接交流。名字和例程都是描述它们功能的缩写，所以注释没有必要，只会把代码搞得支离破碎。如果有一个结构我打算用在五六个地方，我也许会在注释里描述一下此结构的各个组成部分。除此之外，我从来不在程序里放注释。

我阅读代码时需要有对于细节的良好记忆，虽说我的记忆力总体说来并不好。我记不住别人的生日。我知道我在某个地方上过高中，但具体我一点都不记得了。对于那类东西，我的记忆力非常糟。但是，如果给我一个疑难问题，比如做这做那会发生些什么，我大概可以把代码全记住，并可以很好地猜出代码运行的结果。记忆一定是具有选择性的。我知道我的记忆力有时候都会让我自己吃惊——不管是好的方面，还是不好的方面。



采访者：*你写的源代码采用特殊风格吗？*

罗伊森：我想每个人都有自己的风格。我的代码很散，里面有大块的留白。我总是让不同例程需要用到的函数彼此独立，来节约代码、提高速度。

采访者：*你认为有开发程序的唯一正确方式吗？*

罗伊森：当然没有。人们得遵从他们自己的工作方式和习惯。我通常坐得离终端很远，脚放在桌子上，够键盘时就好像在画架上作画一样。这是我的选择，并不意味着其他人不可以坐下来像打字一样盯着键盘。设计好程序的唯一必要条件是兴趣。很显然，如果你想把一件事做好的话，就需要有很大的兴趣。如果你不上心，纵然你有盖世之才也只会得到糟糕的结果；如果你真的上心，只需三分之一的才能就能弄出一些不错的东西来。

采访者：*是什么使你离开了世界银行，开创自己的公司？*

罗伊森：我的理念是在什么情况下都要尽力而为。我那时感到了灰心，因为

看起来好主意没人实行，时间都浪费在了开发糟糕的点子上。世界卫生组织和世界银行都是如此。

采访者：为什么你认为好主意没人实行？

罗伊森：大机构并没有创造环境来鼓励员工进行创造性的思考。员工做到了要求做的事情，就可以得到报偿，多做也没有用。没有动力的话，给程序员提要求的人拿不出引人入胜的项目，而程序员也不会花时间去思考解决问题的有趣方式。在我工作过的机构里，很多时候，我们甚至根本没有去解决什么问题，因为我们首先要做可行性分析，其费时往往超过真正完成工作所需的时间。

我父亲老是说："你的直接主管如何，你的工作就如何。"这对我来说完全正确。当我有好的直接主管时，他们给我充分的自由，我工作时很愉快，晚上回家时也很高兴。但如果我遇上的是糟糕的主管——某个对项目毫无兴趣的家伙，或者彼此之间没有任何共同兴趣——工作就会很糟糕。



我退出的主要原因是我厌倦了每晚回家一无所成的感觉。我想要的感觉是我可以给家里挣钱。然而，实际情况是，我每晚回家时会说："嗯，我今天受够了，所以这钱是我该得的。"这可不是健康的生活方式。

采访者：你做了什么让生活更健康？

罗伊森：啊，我买了一台计算机，开始在晚上搞我自己的点子。我在银行里有大约6000美元，本来打算和妻子一起买辆新车或是可编程计算器上。个人计算机在那时候才刚开始出现。我越考虑该买什么，计算机就越显得是该买的。所以我就买了台48K带终端的CP/M机器。我白天仍然做原先的工作，晚上和周末则在家干自己的活。我不是因为要从这些点子获利而去进行开发，而是因为我想要看看它们是否有效。我想，我有些点子还是很有价值的。

采访者：那就该是T/Maker的起点了。你从哪儿得到的灵感？

罗伊森：那时候，我为预算部做了很多的编程。预算部的主要工作任务就是把很多行列加起来。他们会给我们带来很多需解决的问题，我们则会用像Focus这样的他们理解不了的大数据库打发走他们。从长期来看，他们没得到想要的报告，却有了大堆使用分时机器的账单。这是巨大的资源浪费，就像用高射炮打苍蝇一样。

所以我试着创建一种简单的电子表格，易于学习，给用户他们想要的结果。我希望它可以很容易计算总计和小计，还可以让用户在屏幕上的任何位置输入文本。一开始，我在世界银行做这个程序，把它设计成在UNIX下实现，那样人们可以无需使用大型机来访问数据表格。我想要改进这个程序，但是世界银行里没人有兴趣。然后我去好几家公司想要说服他们把这个程序加到他们的分时系统里去，没人有一丁点儿兴趣，每个人都认为我是个傻瓜。

采访者：但你没有放弃？

罗伊森：我得出结论，我要有进展的唯一方式是使用我自己的机器。这对我很有风险，因为那时我的印象是，你得是个大公司，有很多资源，组织起人力，才能启动一个像我的电子数据表这样规模的项目。我认为小公司真的成功不了。我花了一年时间，才让产品的第一版跑起来。

采访者：在那一年里，你有没有担心过竞争——有人在产品上把你击败？

罗伊森：VisiCalc在我们完成T/Maker之前约6个月就发布了。那真把我吓坏了，因为它听起来就像相同的产品。我甚至不想去看一下，因为我非常确定它会和我们完全一样。我记得我是直到完成了T/Maker之后才去看的VisiCalc，然后我才意识到它完全不一样。当然，VisiCalc赚到了几百万，而T/Maker没那么多，但我知道这并不一定意味着产品的质量上有什么问题。



采访者：你可以描述一下T/Maker电子数据表和VisiCalc哪些地方不一样吗？

罗伊森：T/Maker用了完全不同的方式。我们的程序使用一个空白的屏幕，而VisiCalc把用户的屏幕分成了很多的单元格，你可以在每个单元格里输入数字和等式。T/Maker的方式要轻松得多：程序的计算部分隐藏起来了。用它的时候，你感觉就像是在黑板上写字，而不像是在运行程序。

T/Maker有一个很有趣的特性，你可以把一条信息、一个逻辑思想，竖写而不是横写在屏幕上。我不会把这说成是革命性的概念，但我从没见到过其他程序这么做。

采访者：你设计T/Maker时有什么准则吗？

罗伊森：首先，重点是解决简单问题的方案必须简单。如果你想计算3行之和，你输入数字时要在屏幕左边放3个加号，然后在下面放一个等号，表示

你要在这儿得到一个总和。没什么比这更简单了。我没有试图创造一个一般化的学术解决方案，那样的话会提供过多的可能性，把用户弄糊涂。我选择的方案针对的主要是需要把一列列的数字加起来的人。在T/Maker里，用户可以直接定义满足他们所有规定的整齐的报表，在屏幕上看满意了就能按这样子打印出来。可以这样做的原因是，T/Maker允许人们把文字放到屏幕上的任何位置，这和大部分其他系统都不一样。

另一个T/Maker的重要特性和我的设计风格有关。当让我选择实现一个特殊的有用功能还是一般化的功能时，我通常会选择一般化的功能，前提条件是简单用途的使用体验不会变糟。其他软件包就僵化得多。他们假设用户需要两行的标题，页码放在底部。在T/Maker里，标题的行数可变，页码可以放在页面上的任何位置。这对只想要两行标题的人来说是稍稍难了一点点，但对长期使用它的人来说，程序功能就更强大了。写一个像T/Maker这样的软件包需要做出大量类似的判断，我相信T/Maker里做的大部分判断都很不错。



采访者：开发T/Maker感觉如何？

罗伊森：非常有趣。每次当我开始编译后新功能可以工作时，我都非常激动。有人看了软件，认为已经可以销售了，那就真的让人兴奋了。当然，更让人兴奋的是我开始源源不断地收到支票——我真的可以靠做一件让自己感到满足的事情来谋生了。

采访者：写这个程序时，你碰到的最大问题是什么？

罗伊森：只有一台计算机很麻烦，它老是在我要做产品功能演示的前一天坏掉。另外，我得写一个全屏编辑器，因为T/Maker里使用了可视类型的语法。我以前从没做过这样的东西，所以做起来有点复杂。

采访者：你用什么语言写程序？

罗伊森：最初的版本是用CBASIC写的，因为那时没有其他可用的工具。回想起来，我觉得还不坏。它不但工作了，而且还不算慢。当我最终可以用C（BDS C）的时候，所有东西都快了十倍。直到今天，能在CBASIC里做出一个全屏编辑器，而且速度还能算快，仍然是一件让我非常骄傲的成就。

采访者：在写电子表格程序时，有什么意外发生吗？

罗伊森：和用户接触真是有趣极了。发烧友有时会给我们写信，有些邮件非常感人。有一次我们给一所监狱送了一份免费的T/Maker，有人后来就给我们写信，告诉我们使用这个程序他们有多快乐，以及他们多么感谢有人还关心囚犯。南加州有个人给我们寄来了一箱鳄梨，因为他使用T/Maker来运营他的大农场。这就像有了一个大家庭一样，即使你们从未谋面，只要在电话上交谈过、写过一两封信就可以让用户成为大家庭的一员。这是这个业务附带的美妙作用。

采访者：自从你写了T/Maker之后，生活变化多吗？

罗伊森：我的生活变了很多，特别是我对工作和薪水的看法有了很大改变。当我为其他人工作时，我认为再找工作的话，一定要比前一份工作至少高出5%的收入，而且我很不愿意冒险。我总是想着薪水和下一个发薪日，因为我可能想要买个新电视或立体声音响什么的。当我还在做这些早期的工作时，我常常禁不住想：为什么会有人给我这么好的薪水，而我并没有做什么对世界有益的事——我只是无所事事地和别人一起喝咖啡，写永远不会被实现的可行性报告和没人会去用的程序。我的工作只对我有好处，而且这唯一的好处就是我的薪水。过了一段时间之后，我开始感到自己很没用，因为我没做任何有用的事情。

现在我的感觉则变成了发现自己有一些相当有用的技能。我从作品本身得到满足，而不再老是想到薪水。如果现在给我两个选择，一个是拿一半的钱做我喜欢做的事，一个是拿两倍的钱做别人让我做的事，我会选择低收入和自由。当在工作上很快乐时，我并不需要开一辆保时捷，因为那并不是生活的重要组成部分。做自己喜欢的工作则是我生活的一个重大组成部分。

采访者：你现在在开发什么？

罗伊森：我在开发T/Maker的一个修订版本。我对付这个程序已经5年了，已经看到很多T/Maker可以改进的地方。我发现了当初写这个程序时还发现不了的问题。若没有真正尝试某件东西并大量使用上一段时间，你是看不到里面的漏洞的。你会只见树木，不见森林。T/Maker里的某些树木需要重新排列一下，才会变成更好的森林。

老产品对于不阅读手册或不熟悉计算机术语的用户不太友好。那是我以



前不太留意的地方。因为我每天都用计算机，一用一整天，要记住什么快捷键的话，记住就是了。现在对于那些不常用T/Maker的人，比如只在周末使用的用户，我可以让他们更容易使用程序。因为要让别人星期天用过后隔一周还能记住如何操作是有点难。

采访者：你有没有考虑过做一个和T/Maker完全不一样的东西？

罗伊森：我不介意尝试其他的东西。不过，我已经在T/Maker上投入了这么多的时间、代码和精力，很难放弃它去尝试新的东西。在任何新项目里，都会有一个耗时的启动过程。而T/Maker里也一直可以添加新模块。现在我还有一个程序员帮忙，他对机器的了解比我更深入，所以我们又有些新的事情可以做。

采访者：你认为编程是艺术、科学、手艺还是技能？

罗伊森：我当然认为这是艺术。我认为任何做得好的事情都可以成为艺术，因为任何工作都可以加上一定的品位或艺术天赋来完成。在我自己的工作中，我不但关心代码做什么，也关心它看起来如何。我经常重新整理代码，就为了让它看起来更漂亮。



采访者：你可以拿编程和其他艺术比较一下吗——比如作曲、绘画和写作？

罗伊森：我只了解其中一项，所以很难作出比较。我完全想不出来作曲家创作音乐时会希望得到什么。艺术性在于你有兴趣去做一件事，并努力把它做好。我并不会在写程序时是一个样子，在房间里贴墙纸时又变成另一个样子。

采访者：你认为计算机科学是真正的科学吗？

罗伊森：我不太相信学习计算机科学可以造就好的程序员。显然，你必须学习如何开发散列表以及类似的算法。不过，自己学有可能更好，就像我一样。当然，我只能自己教自己，因为那时还没什么参考书和课程。我这么说，是因为课本或课堂解决不了大部分实际发生的问题。你不能翻到某本书的第5页，找到问题某一部分的解决方案，然后到第6页找到下一部分的解决方案，最后把两部分拼起来，整个问题就解决了。这样的解决方案从来没有效果。

任何问题的最大也是最困难的部分就是决定采取什么方向。当我摆弄某件东西足够长时间之后，我会搞明白计划该是什么样，然后实际编程就不怎么难了。在计算机科学里学上10种不同的排序算法对于设计一个合适的解决

方案并没有什么帮助。

采访者：在这里，你给了新一代的程序员一些建议。有什么人对你有特别影响吗？

罗伊森：没什么特别的人。我倒很喜欢*Zen and the Art of Motorcycle Maintenance*[①]一书。虽然里面差不多有四分之三我不理解，但我能理解的有些段落给了我很大的冲击。在读那本书时，我开始更多地思考质量问题和如何把事情做好，即使看起来没人关心这个问题。作者描述了一个摩托车手把车子拿到店里去修，修理工想都没想，就抓起一件工具，错误的工具，一下就弄坏了一个螺母上的螺纹。你发现有些人就是这个样子，把工作当作无需思考的活动。作者描述了把事情做好所带来的快乐，这对我影响很深。

采访者：你和其他程序员交流多吗？

罗伊森：不太多。我更愿意实行自己的点子。我不了解其他程序员是如何工作的，但我认为他们对我的想法没什么兴趣。我当初推广T/Maker这个点子没有成功，也确认了这一点。

编程的一个伟大之处就是你可以拥有设备自己来干活。要完成某件工作，你不一定需要别人跟你合作。你可以非常独立。在公司里，我已经把去办公室的次数减少到了约一周一次。我不喜欢那些必须在办公室里完成的工作，像规划市场战略、坐下来开会什么的。那不能给我带来满足感，我更喜欢一个人在家工作。

采访者：你认为什么可以造就一个好的程序员？

罗伊森：好的程序员是一个对他所从事的工作有相当兴趣的人。他的目标是做出一个尽可能好的产品，就像军队宣传片说的那样，无论多难也要完成目标。他也需要对细节有良好的记忆。我可以拣起一段我一两年前写的代码，一般情况下我很快就能理解当初的意图。

采访者：如果一个程序员写了一个优秀的程序，它是不是一定会成功呢？

罗伊森：我过去是这么认为的。但现在我真的不确定了。这取决于你如何定

① 中文版名为《万里任禅游》，重庆出版社2006年出版。



义成功。我对成功的理解是，白天可以做自己喜欢的事情，月末可以付得起账单。这样的成功不难获得。基本上你做任何事情都可能达到这样的成功。但是，我也知道，即使写出一个好10倍的软件包，你也取代不了像Lotus这样非常成功的产品。不仅如此，好的主意是很难获得的。不是说你坐下来说“我今天一定要有一个好主意”就能成的。也不是说你拉10个人过来，就能想出50个点子。我这一生也只有三四个点子令人满意，可以开发成有用的产品。其中的一个就是T/Maker，至于其他点子，因为没有去实行它们，现在我连它们究竟是什么都想不起来了。

采访者：*在接下来几年里，你认为计算机行业会发生些什么？*

罗伊森：我不知道。我可能是你最不该问的人，因为我不跟潮流。当我拿起一本个人计算机杂志翻看时，我总是感到很沮丧。里面有那么多吸引人的产品。当我看到那些产品闪亮的屏幕截图时，总禁不住会觉得我们自己的产品很差劲。对付这种沮丧的一种解决方案就是不去看那些杂志。这样做的结果就是我并不熟悉别人在做些什么。但只要我们有顾客，我就不怎么担心。我们不断地改进T/Maker，他们也继续购买更新版本。我们有一些相当稳固的整机厂商客户，只要我们不时地提供更好的版本，我想他们就会继续和我们做生意。



采访者：*你认为计算机有一天会变聪明吗？*

罗伊森：计算机非常复杂，非常快速，也非常愚蠢。对付所有费时而又枯燥的工作，用计算机实在是太方便了。计算机可以完成这样的工作，因为它不会感到厌烦。它不会在早晨起床时说：“我讨厌做加法，我不想再干了。”计算机会一直做。它像螺丝刀一样只是件工具。如果使用得当的话，你可以让自己少干很多乏味的活。

对于很多任务，使用一定的算法会比单靠直觉得到更好的结果。举例来说，对于扁桃体炎或某些骨折之类的小病，计算机可以比医生更快更好地作出诊断。需要速度的应用会显得特别引人注目。医学诊断会让计算机显得很聪明，因为计算机很快而且不会犯逻辑错误。但是，如果有什么事情需要深入的思考——比如某些癌或者病毒感染之类的疑难杂症的诊断——我们并不知道能解决问题的全部思路，我想计算机也不太可能会有天赋异禀能解决

此类问题。

采访者：你关心计算机在社会中的作用吗？

罗伊森：不太关心。对于整个行业或者整个世界，我都不那么关心。对于世界，我的关心程度只限于我希望世界是安宁的，我也希望尽我能尽的本分让世界安宁。就我的工作而言，我不关心我的程序的营销战略是否很好，计算机行业是否会经历衰退，或者现在是否是时候做某个特别项目。对这些我毫不关心。因为人如果花上一辈子的时间去想是否该做某件事，到最后就会没时间真正完成一件事了。这有点像交税。我乐意交税，因为这是对社会的贡献，而没有社会的话我也赚不了钱。我知道有些人时常为交税烦恼，在烦恼上花的时间都可以用来赚很多钱了。

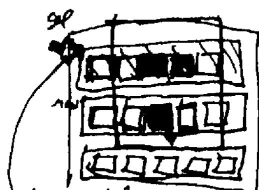
续写传奇人生

当时专注开发T/Maker（T/Maker的合伙人和CEO是他的妹妹海蒂·罗伊森），1986年的时候，他的妹妹和另外一位合作伙伴罗尔·法罗斯买下了这家公司并继续运营。到1993年，T/Maker的各种版本加起来已经卖了25万份。

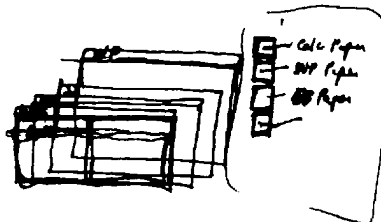
罗伊森一直都是SOHO一族，他选择在家做自己的项目，或者帮助企业开发软件。相比全职上班，他更喜欢这种状态。他的网站上列出了一些做过的项目，例如I Hate Algebra（我恨代数）电子表格是他第一款Windows软件、Chartbook System是为股票交易公司开发的系统。他还为第一家在线冲印店iPrint.com设计过网站，直到现在还在维护和支持该网站。他自己一生钟爱拼字游戏，终于将其搬上电脑，设计了WildWords这款游戏。

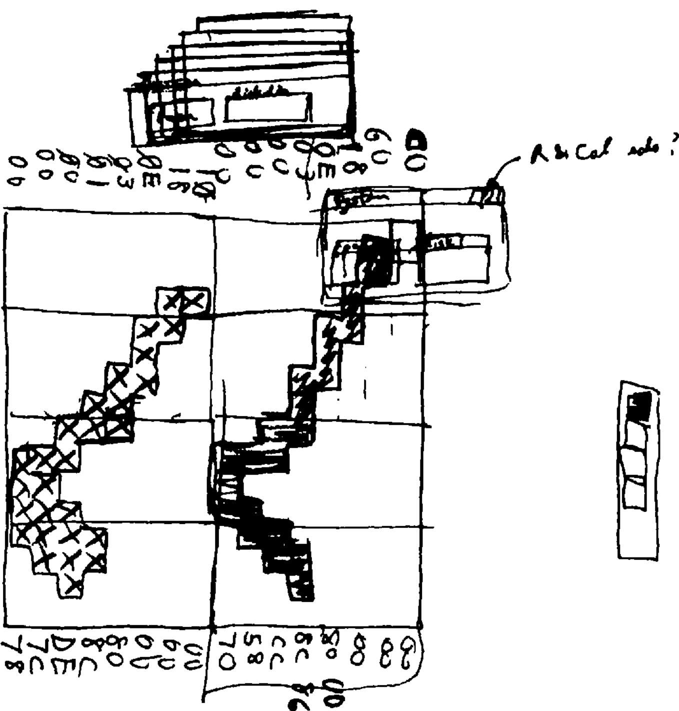


How to scroll spread sheets



the problem is Fred currently forces all t.f.'s in the path to be visible.





卡尔用这些“电子表格漫游”来帮他设计Framework软件。更多的草图和笔记，请参看附录。



13

鲍勃·卡尔



作为Ashton-Tate公司的首席科学家，29岁的罗伯特·卡尔（Robert Carr）负责新产品和新技术的研究工作。作为集成化软件领域的开拓者之一，卡尔是Framework软件及其后续产品Framework II软件的设计师和主要开发者。

在此之前，卡尔曾担任Forefront公司的董事长。1983年7月，他与人合作成立了Forefront公司，为的是能完成他自己于15个月前就开始了的Framework软件的开发工作。Ashton-Tate公司同意以注资Forefront公司的方式来获得Framework软件的市场开发权。1985年7月，Ashton-Tate公司收购了Forefront公司。

在开发Framework软件并共同创建Forefront公司之前，卡尔还参加了最早的集成化软件包之一的Context MBA软件的开发工作，担任其程序开发顾问。在此之前，卡尔曾在施乐PARC从事施乐之星及Smalltalk前身产品的开发工作。他同时拥有斯坦福大学计算机科学的学士学位和硕士学位。

Forefront是个小公司，有一个核心的程序员小组，他们共同工作、相互合作。鲍勃·卡尔年轻、充满活力、谦逊而又精明，是这个核心小组的技术主管。

卡尔穿着整齐，戴着玳瑁壳眼镜，穿着灯心绒裤和牛津布衬衫。谈话间，

他一边做着手势，一边在白板上画图，侃侃而谈，深思熟虑，条理清晰。他以他最成功的项目Framework软件为例，回忆着自己的过去和他的程序设计方法。

*　　　*　　　*

采访者：你从一开始就打算成为一名程序员吗？

卡尔：在斯坦福上大学时，我最初喜欢的是人文科学，但我这人比较实际，所以又倾向于学工程专业，那时因拿不定主意而备感苦恼。我内心游移不定，因此把上课的时间分出来同时修这两个专业的课程，以便在这两者之间做出选择。

我喜欢写作，觉得自己文笔不错。我现在仍幻想着有一天能当个小说家。用英文写作、编写软件决不仅仅是在纸上搞些代码或写点儿东西，而是可以极大地促进思维进化的过程。在被迫把想法写下来的时候，思考得就会更深远了。



到大三过了一半的时候，我做出了折中的选择。我说："我知道了！我喜爱写作，那就要以此为生，可我又那么实际，那就做一个技术撰稿人吧。这样既有饭吃，又能搞些发明和创新的写作。"我咨询了几位新闻学教授，他们指出，要想获得一份好的技术写作工作，没有什么比拥有一个技术类的学位更重要的了，所以我决定去学技术专业。但随后我却发现自己并不喜欢工程：机械工程——太讨厌了；电气工程很有意思，但我却无法驾驭它；化学工程也别提了。没有一个是我想学的。

在那个学季，我上了一门初级编程课。为了使用ALGOL W语言编程，必须在大型机上使用穿孔卡。我很喜欢这门课。在那个学季我还找到一份工作，为斯坦福大学一个叫LOTS的新分时系统做晚间保安。那个系统用的是DEC 20机器，带30个终端，那个地方通宵开放。我坐在前台，以确保设备不会被人拿走。

我本来是无法接触到计算机的，但是感谢拉尔夫·戈林（Ralph Gorin），他在保安的办公桌上放了一台终端，这样我就有机会学到了字处理，并开始一些简单的编程。在此之前我从未接触过计算机，那会儿我马上决定去攻读计算机科学学位。不过有一个障碍，斯坦福大学并没有设立计算机科学学士学位。但我是不会被官僚制度所阻止的。我修了一个名为通用工程的工程学位，在那里你可以设计自己的专业或"发展方向"。我选择了计算机科学作

为我的专业，并称之为计算机工程。在经过接下来的四个学季的学习后，我获得了学士学位，非常兴奋和开心。而那时我也清楚地认识到，我更愿意编程而不是去编写技术文档。

采访者：于是在1978年，你在获得大学本科学位后就进入了专业编程领域？

卡尔：是的。我运气太好了，当时正好赶上施乐PARC从主研究中心拆分出来一个叫做先进系统部（ASD）的分部。ASD的办公室就在斯坦福大学的隔壁。当时，施乐公司在许多方面都处于世界领先水平，先进系统部（ASD）则是一个从事研究活动的温床，拥有不少最优秀的人才。比如有查尔斯·西蒙尼；有像本·韦格布雷特（Ben Wegbreit）和杰·斯皮特森（Jay Spitzen）这样的哈佛教授，他们俩曾帮助创办了Convergent技术公司；有道格·布罗兹（Doug Brotz），现在是Adobe系统公司的核心程序员之一；还有创办了GRiD系统公司的约翰·艾伦比（John Ellenby）。

采访者：施乐公司成立先进系统部的目的是什么？



卡尔：对于如何把非常好的技术开发成果转化到产品开发阶段，施乐PARC存在一些问题。设立ASD的目的就是使这些先进的技术可以转化为现实世界的系统，但又不是最终的市场产品。ASD是一个介于初始开发工作部门和位于洛杉矶的最终产品部门之间的中间人，比如说，施乐之星是在最终产品部门生产的，而ASD则开发了原型来试探市场前景。

我是一个基层程序员。我所在的小组开发了集成化的办公系统，执行文字处理、数据库和以日历为导向的作业。所有这些都建立在单个的数据库之上。

采访者：你在施乐和ASD的工作经历是否让你学到了有关管理及有创意的工作环境方面的东西？

卡尔：学到了，从那时起，特别是在这儿——Forefront公司，我总是试着结交比我棒的人。我为Forefront公司聘请的很多人都是比我优秀的程序员，我从他们身上学到了很多。ASD也让我了解到优秀的软件开发是由一个良好的初始设计加上其后一系列非常扎实的工程而组成的。

采访者：ASD最后怎么样了？

卡尔：施乐公司经历了重组，ASD解散了，并且重新并入到施乐公司的其他部门。当时很多人离开了。我回到学校完成我的硕士学位。我对计算机科学

很气馁和不满。编程完全无法发挥我的创造力。

我觉得自己像是池塘里的一条小鱼，周围全是非常大的鱼。一方面，这是一个绝佳的学习机会，但另一方面，我没有任何自己创造或设计的机会。在那里你必须做个学徒，而我却又不是那么心甘情愿。

完成硕士学位就成了一条简单的出路。后来我决定和过去的生活一刀两断，来个全新的开始，看看自己到底想干些什么。我把车卖了2000美元，去了拉丁美洲。我一直觉得生活在外国、学习另一种语言会是一件很好的事情，所以我想去旅行，在国外的什么地方住下来，找个工作，可能就是做一个程序员吧。

我本打算一直去到南美的，但却没能走出墨西哥，因为我真是太喜欢那儿了。在墨西哥城，我曾在一家青年旅社做了六个月的义工，就和开这家旅社的夫妇住在一起。那段时期结束时，我觉得已经准备好了，可以再次工作了，所以就又回到洛杉矶找工作。



最后我找到了一份临时工，在可编程计数器HP 41C上编程。我所写的软件是为了分析建立一个集中式的炼油厂对我的客户——一个石油大王——来说是否会有利润。在做这份工作时，我发现附近一个新成立的公司正在招聘程序员。那就是Context管理系统公司，当时他们正在开发Context MBA软件。

采访者：Context MBA软件是不是VisiCalc和Lotus 1-2-3软件的竞争者？

卡尔：Context MBA软件的构想是一个更引人注目的新产品。它是一个单个的程序，有着巨大的而强悍的报表功能，远远超过了当时市场上最畅销的电子表格软件VisiCalc。它可以在电子表格、图形和字处理程序里做数据库或数据管理操作。它跟Lotus 1-2-3软件非常像。

在我工作了差不多一年后，软件开发完成并于1982年7月交付使用，比Lotus 1-2-3软件要早上市半年。公司曾对该产品寄予厚望，希望它最终能够像Lotus 1-2-3那样，也就是说，取得超乎想象的成功。但Context MBA软件却并没有获得成功。由于一些显而易见的原因，Lotus 1-2-3在问世后抢占了我们的风头。虽然他们产品的功能并不如Context MBA软件丰富，但速度却快得多。Lotus 1-2-3软件占据了绝对的市场优势。Context公司认为会有许多不兼容的PC硬件，可移植性将是一个关键的竞争优势，所以把希望都寄托到了这方面。他们采用了UCSD操作系统，那是一个可移植性非常好的环境，但速度却十分慢。因为IBM PC设置了标准，使Context公司失去了本来可以

获得成功的机会。用汇编语言编程并为IBM PC量身定做才应当是制胜之道。于是Lotus 1-2-3软件胜出而Context却歇业了。

采访者：*是否可以说是Context的工作经历重新点燃了你对编程的兴趣？*

卡尔：再次进入计算机行业确实激发了我的热情。在Context公司当软件副总裁的感觉非常好，我得到了很多软件方面的培训，都跟手头有待解决的问题密切相关。所以在Context公司，突然间，我更像是池塘里的一条中等大小的鱼了。我能够有所作为了。这个经历让我非常有成就感，也让我知道自己能够享受计算机编程了。

同时，我目睹了这家公司在一年的时间里从4个人到25个人的发展史。我目睹了创业的过程。那是许多、许多的决策和任务，由每一个人用他们自己的时间一点点地完成，是一步一个脚印发展起来的。

采访者：*在早期，和你一起工作过的人曾对你的编程风格产生过影响吗？*

卡尔：大约有三个程序员让我学到很多。首先是克拉克·威尔考克斯（Clark Wilcox）。我在斯坦福大学的时候，他正开发一种叫做Mainsail的语言。他是一个非常多产的程序员，在编程上一直非常严谨。如果他写了一条开始指令，他会马上在下面的几行后面加上一条结束指令。那时，以我的思维方式，觉得那不是找罪受嘛。然而，我现在却认为这个习惯对于锻炼技能是非常重要的。我现在会一丝不苟地照做。

接下来是在施乐公司时的杰·斯皮特森和Context公司的吉姆·皮特森（Jim Peterson）。从他们两人身上我学到了很多软件设计的知识。他们负责架构方面的工作，设计了部分架构，而我的工作是实现这个架构，所以我从他们所做出的判断中学到了很多。

采访者：*Framework软件的想法是什么时候产生的？*

卡尔：我与Context公司的合同快结束时，开始考虑向用户提供一个多功能系统。我也开始认识到，以一个巨大的电子表格为基础构建所有这些功能的做法是错误的。

有一天我在想，如果打开一个电子表格单元，发现里面还有一个电子表格，再打开发现里面还有另一个电子表格，那会怎样呢？我画了一个简单的图来表达这个信息嵌套的概念，就是长方形里套长方形再套长方形。后来有一天我跟一个朋友说，虽然Context MBA是一个令人振奋的产品，但我总觉



得可以有更好的方法来实现。然后他问了我一个非常简单的问题："嗯，那你会怎么做呢？"这个问题激发了我的热情，并且成为Framework软件的催化剂。

我需要考验自己。我对生命中曾经的几个未完成的项目感到难过。从那时起，我那个关于更好的解决方案的模糊感觉就变得思如泉涌，并汇成一个单一的概念，那就是长方形里嵌着长方形，窗口里套着窗口。

我每天早上和晚上做着Framework软件的构想，只是随手写写画画。在白天我照样编程。经过四个月在纸上的头脑风暴后，我有了研究方向。我和Context公司的管理层谈了我的构想。他们很兴奋，但他们已经有了对公司和产品线的发展计划。我能够实现梦想的机会微乎其微，所以我离开了Context公司，搬到了旧金山，在那里我就不会受到诱惑，再回去为他们工作了。要知道他们都是很棒的人。我依靠积蓄生活并继续着Framework的设计，想着如果事实证明这是一个可行的方案，那就有可能变成一个商机。我被迷住了。我的思绪奔驰，以从未有过的努力工作着。



采访者：你当时做Framework软件研发的目标和工作规则是什么？

卡尔：我得到的一个忠告就是尽可能推迟编码。一旦写了一堆代码后，就很难改变方向了。它就好比混凝土一样成了阻碍。因此，我尽可能推迟编码，但程序设计却一直在我的脑海中，从没有停止过。

原本我对Framework软件的设想是一个包含字处理、电子表格、图形和数据库等在内的多功能产品。后来，我又雄心勃勃地想实现引导程序的概念，换句话说，在底层实现一个系统或是一种语言，系统的其余部分都可以在这个基础上构建起来。我想做出一台神奇的、时髦的机器，可以作为Framework软件的语言和系统。但我相当沮丧，因为我无法驯服那个怪物。所以我缩减了原来的设计。

我的另一个目标是，通过实现一个单一的通用数据对象来尽可能简化核心设计和体系结构。这样做的原因是，在一层又一层地添加详细的功能时会占用内存，所以在内部需要一个特别简单的格式，否则程序注定会是一头大肥猪。

我最终放弃了实现一个底层语言的目标，并选择用C语言来编码。很可惜的是，那个C语言编译器真是太烂了，我很快就发现它根本不能用。所以我决定用汇编语言来编码，因为汇编语言在8086体系结构上对内存的处理有很大优势。

这些一层层的圆圈代表着卡尔在开发Framework软件期间的“向结构化的最初探索”。

1983年1月，我开始按我的设想来编程。那是一个单一的数据对象，我称之为一个“框”，基于这个“框”，整个系统都可以实现。那真是一场赌博。如果可行，我就可以在很短的时间内做出一个系统让它运行起来。如果失败，我就死路一条了。

结果它大获成功。在短短几个月内，我做了很多用户界面的工作，在半年内，我得到了一个非常有效的工作原型。虽然它只完成了我的设想的五分之一，但这正是我一直想要的系统。它提供了桌面、内部相互嵌套的框架、字处理、文件归档、打印功能和下拉菜单。虽然没有提供语言，但你可以创

建电子表格、建立数据库并进行分类。在六个月后，我依靠一己之力就实现了所有这些功能。

采访者：请再说说你的单一数据对象的概念，也就是你所说的“框”。

卡尔：有各种不同的“框”，“框”又具有不同的属性。比如，一个“框”的属性是横向的，另一个“框”的属性是纵向的。这样，如果把一些横向的“框”连起来，并在里面插入一个纵向的“框”，就做成了一个电子表格。电子表格和包括数据库在内的其他功能一样，都是利用同样的构件开发出来的。于是，我们可以一次又一次地反复利用同一段代码，代码可以重复利用，功能也可以重复利用。

例如，Framework软件系统中所有的字处理功能都可以在电子表格中使用。为了在一个窗口式环境中实现这些功能，必须建立一个200K的电子表格模块，并在另一个窗口中建立一个文字处理模块。如果需要电子表格具有文字处理的能力，就必须添加额外50K来做粗体、斜体、换行、查找、替换等字处理的功能。



采访者：当Framework软件逐步变成最终产品时，出现了新的功能吗？

卡尔：我们增加了新的功能并使产品更加健壮。我们和Ashton-Tate决定添加许多全新的功能。最大的新功能应当是在两个方面：一是在开始的时候并没有打算做成一个功能完备的电子表格，但后来却成为了这样的电子表格；二是我们增加了在通讯方面的能力来与莲花公司的Symphony软件竞争。

最好的设计工作是开始时有一个精良优秀的设计，让它运转起来，然后对只有在使用过程中才能想到的其他想法做出响应。这种方法成本高昂。通常要重写很多代码，或者有可能要抛弃整段代码。我把做这种任务的那些程序员称为“英雄”。例如，做Framework软件时，在其他程序员加入之前，我已经做了一个字处理器。丹·阿特曼（Dan Altman）来了后，在不到一年的时间里，硬是把字处理器程序重新改写了三遍。

在开发Framework II时，他又做了重大改动，再次重新改写了字处理器程序，以便更有效率地利用内存。当然，他的工作是有报酬的，但是，当你已经付出了最大努力，却又要回去从头重做时，你所要付出的努力，可就不是仅仅完成本职工作所能相比的了。这就是为什么我把这些程序员称为“英雄”的原因。

采访者：既要创办公司，又要开发产品，这是什么样的情形呢？

卡尔：很艰难。我想我们的境况比起大多数公司来说已经算是好的了，因为我们公司只关注软件开发方向。我们的宗旨是为程序员创造一个理想的编程环境。公司并不完美，但这里所有的人都可以作证，包括我在内，这是我们所干过的最好的工作。

同时也很可怕，因为我总是担心失去我们的环境。公司发展得越好，我就越担心它什么时候就要结束了。在并入Ashton-Tate公司后，对我们而言是有一些变化的。从前，我和我的伙伴不得不操心发工资、医疗保险、办公场所、可以承诺租赁多长时间等，我们还不得不担心与Ashton-Tate的合同。我们会得到足够的钱吗？他们会起诉我们吗？我们会起诉他们吗？我们是否会成功？如果Framework软件失败了我们该怎么办？当你自己开公司时，所有这些就都是重要的问题。合并后，我们从所有这些问题和忧虑中解脱了出来，所以现在，我需要关心的又只是有关软件开发的问题了。



采访者：这么说，正是与Ashton-Tate公司合并的决定使你可以重返编程？

卡尔：当我们还是一个独立的公司时，那么多管理问题让我没法编程。把我从中解放出来，让我可以有时间编程，的确是合并的原因之一。在最初的Framework软件开发中，我是核心成员。我写了一年的代码，然后把更多的时间花在了设计上。接着有五个程序员和一个文档设计师加入到这个项目中，我们夜以继日地工作了十个月。但是，在开发Framework II时，我并没有在一线做编程工作，而是更关注我们Forefront公司的发展方向。

采访者：请描述一下程序员的“理想环境”是什么。你在Forefront开创这样一个环境时的角色是什么？

卡尔：工作非常重要的一部分是我们之间的相互合作，根据达成的一致意见进行设计，这意味着团队必须要小，也就是说，不能超过七八个人。我们紧密合作，不断更新设计，在白板上讨论来、讨论去，而且不会有哪一个人是独断专行。在团队讨论被卡住时我们也会有一个仲裁者，但这种情况并不多见。

我的角色是个促进者，画出我的设计构想，帮助开发小组得出结论。那不是我的结论，而是由整个开发小组逐渐形成的。

我们尝试着实施一个构想，观察其带来的反应，以此得到一个新的、

更好的构想。在团队相互合作的动力中，需要有人能擅长勾画出新的想法，然后激励大家来一起完成。偶尔也有无法取得共识的情况，我们会回过头去看看，是有什么时间上的限制、资金上的限制或是空间上的限制吗？再从那里重新出发做出决定。原始Framework软件的进程是非常明显的迭代和演进风格。

采访者：听起来好像你更重视编程过程而不是业务方面。

卡尔：是的，我喜欢编程，但更重要的是，我喜欢集中精力做事情。我无法把时间分给四个不同的项目，而每个项目都要求我百分之百地关注。如果时间只用在一个项目上，我会发挥得更好。既要发挥技术带头人的作用，又要帮助经营公司，我的时间被一分为二了。

采访者：你用的这些技术是在哪里学的？



卡尔：来自以往的经验和自我反省。我倾向于把事情简化到一个相当简单的框架——如果你不介意使用框架这个术语——然后再把它扩展开来应用到现实世界中。我不苛求完美，并不指望框架对每个应用它的功能来说都非常适合，但我希望不论是什么主题，这个框架都能为使用者提供深刻的见解和指导。

我的数学非常糟糕，但是如果做一个物理学家，我会很高兴的，因为我认为物理学家和我做的是一样的：他们把物理现实简化成为数不多的定理和规则，然后用它们去预测和解释物理现实。我在软件上做着同样的事情——努力提出一些定理。

采访者：你觉得遵循这些技术能够造就优秀的程序员或优秀的程序吗？

卡尔：是的，我觉得计算机科学手艺的关键就是找出规则。我称之为手艺，因为它确实还没有成为一门科学。经典的计算机学科中体现了一些已经发现的主要规则，而你最优秀的程序员通常是那些知道这些普遍规则的程序员。

采访者：你将编程看作是一门手艺。你为什么认为这是一门手艺，而不是一门艺术或是科学？

卡尔：是两者的结合。我非常确定地做出这一论断。当然，软件中有一些是科学性很强、有根有据的规则，它们对于软件开发是至关重要的。但优秀的软件并不止于此。比如说，如果你看看那个产生了Framework软件设计的决

策树的草稿，会看到我花了很长时间在那儿思考、写写画画——许多的潜意识活动。（请查看在本访谈开始处的草稿。）这就是艺术所在。在艺术上，你无法阐明如何达到最终结果。从艺术的角度来看，最好的软件来自于直觉的范畴。

采访者：撇开直觉不说，你在程序设计中是否发掘出了一些具体的规则？

卡尔：首先，做事情的粒度是非常重要的，粒度应该灵活。这个规则可以一次又一次地在世界中发现。在投资上，它表现为多样化的规则：你不会把所有的鸡蛋都放在一个篮子里，同样也不会把投资遍布在三千多个不同的地区，让你无法聪明地管理它们。对于软件而言，用户必须要能够把他们的工作和程序拆分成独立的块，而不是作为单个的巨大实体处理。

第二，粒度遇到的另一个规则，是同质性。软件的体系结构应当是同质的，不应当有很多例外或特殊的情况。如果你设计的系统既能有效地处理大到兆字节的对象，又能有效地处理小到几个字节的对象，那么这个系统最终会变得非常复杂。若只就其中的某一种对象来处理的话，系统将会更好，这就是我所说的同质性。

另一个全局规则是处理递归——它是计算机科学引入到软件的三个最辉煌的、最强大的手段之一。最强大的软件通常在内部采用递归，并常常为用户提供递归功能。

内存管理是一个程序内部体系结构最重要的方面。在这方面的一个原则是永远不要把同样的数据复制两次。包括Context MBA软件在内的一些程序中，同样的数据反复出现了两次，不同的代码模块都在复制同样的数据并将其放置在一个缓冲区中使用，最终是这些数据被散落在整个内存中，到处都是。说它是个弊端，有几方面的原因。因为需要用到所有这些临时的缓冲区，所以对内存的需求会膨胀。但这还是最小的弊端。它还会延长开发时间并增加开发成本，因为它需要更多的时间来开发和调试。这一弊端最讨厌的问题是，当你把数据从一个缓冲区中推到另一个缓冲区时，最终会失去灵活性和功能性，因为当数据在一特定的缓冲区时只能对它做一件事情，如只能在字处理器拥有的缓冲区中进行字处理。与此相反，如果数据的展示和地址只有一个，就会更灵活。

采访者：当你设计用户界面时有没有要遵循的规则或准则？

卡尔：要实现的一个重要特性就是实现我所说的透明度。用户应当忘记在他



们和信息之间还有一个程序。事实上，随着对软件的日益习惯，他们心中应当只会想着手头的任务，而不应当停下来考虑要用什么命令。命令及其行为的一致性和可预见性是增加透明度的两个最重要的因素。

采访者：你是如何做到让Framework软件的用户界面在简明的同时又保留了丰富的功能和性能的?

卡尔：我们制定了菜单方案，说明了约束和规则。我们都知道，过度的感官享乐是祸而非福，对菜单来说也是如此。它们需要很好的规则。

菜单是向用户呈现命令选项的最好方式。但有些菜单系统的分支太多，变得很怪异。还有一些菜单不那么畸形，但也非常复杂，很难学习其使用方法，主要还是因为这些菜单的选项太多了。

在Framework软件内，我们严格执行菜单规则，最终只有九个菜单和四个子菜单。我们使用下拉菜单技术，这样你总是很清楚地知道在菜单的哪个地方。我决定使用下拉菜单而不是Lotus 1-2-3那种风格的菜单，因为在Lotus 1-2-3中你不知道当前的菜单来自哪里，新的菜单覆盖了旧的菜单。而下拉菜单虽然独占屏幕的空间，但用户总是知道自己在哪里。我们所有这些菜单只有100条命令。 而Wordstar已超过130条，Lotus 1-2-3有300条，Symphony 有600条。



采访者：你是以牺牲功能来换取简单化的实现吗?

卡尔：设计中的每一个领域，从建筑到家具，都会确定一些原则，其中最简化具有非常大的好处和优点。在几个特定情况下，我们牺牲了功能；但整体而言，并没有牺牲功能。Framework在单个命令中提供了一组最重要的、有部分重叠的功能。我制定了十来种操作作为所有应用程序的基本操作，如复制、移动、删除、创建、查找等；我认为这些操作，比如复制命令，对字处理器和电子表格都应当是相同的。

为了给不同的领域提供功能，我不得不借用了源于施乐公司的一个很好的设计理念：所有指令都应当作用于用户已经选择或标记的数据。这就是所谓的面向对象的动作设计，而非面向动作的对象设计。

一个面向动作的对象设计，当你按下复制命令时，软件会提示："你要复制什么？"在你标记一些文字后，它会问："你想复制到哪里？"然后复制到那里。这种设计完全是提示驱动的，把用户放在一个交互模式中，在某种程度上是成功的，特别是在指导新手用户上。

而我们的一条面向对象的动作设计的命令作用上往往是相当于其他应

用程序的十条指令。传统的字处理器对于复制单词、复制句子、复制字符等有几个单独的命令。但我们只有一个复制指令，因为Framework软件有一个强大的选择功能，可以非常有效地使用光标指令来选择。

采访者：你觉得有没有其他领域，如心理学或平面设计，可能会对软件设计有帮助？

卡尔：我认为有帮助的一个领域是语言学。我们需要多看看这个领域，看看它能够向软件设计提供什么。目前，我们的产品使用了加密图像、图标或指令菜单上的字符向用户呈现概念、信息或路标。因此，我们需要研究路标学或其他有用的东西。在Forefront公司我们已经开始了这方面的尝试。已经有几个非程序员是有心理学和教学训练背景的，他们为我们的设计带来了巨大财富。

采访者：你会持续地在一个程序上工作，不停地重写、优化和修改吗？

卡尔：虽然一个程序有着无穷变化的可能，但我不会一直不停地改下去。在某些时候，原来的结构或内部体系结构就是因为在它上面反复修改而变得错综复杂了。除此之外，程序返工也太复杂了，效率太低了。

坦白地说，我们真的是聪明些了，可以用新的代码来做Framework II软件，而不会与现存的代码混在一起，因为经历了两年多的迭代，Framework软件已经变得非常庞大了。下一版本，我们会用所学到的经验来构建一个全新的架构，来一个全新的开始， 那将会有效得多。



采访者：你不断提及内部和外部架构。你能解释一下吗？

卡尔：任何产品的外部用户接口都有一个架构或者结构。它有它的路线和途径。程序的内部结构也可以被描述为一个架构。这两个架构在任何程序中都是最重要的。通过观测它们是如何相互联系的，它们是否能够互为映射，你就大致可预测这一程序最终是否能成功了。

最佳程序的外部架构会映射到内部架构中。最好的例子就是电子表格。通常它们在内存中是以单元网格的方式实现的，而用户使用的也是单元网格，所以两种架构完美地互为映射。它们是“一致的”。正是由于这些优点，电子表格可以非常高效、快捷和流畅，也因此在市场上一直非常成功。

但是，如果在电子表格上实现字处理器会怎么样呢？内部已经有一个架构，但是在外部提供的是一个差异很大的结构，一个很长的、像卷轴一样的

滚动文本？你最终所得到的是一个性能不佳的程序，而且用户界面很庞大。实际上，没有哪一个字处理器是成功在电子表格上实现的。

所以我设计Framework软件的一个基本规则就是内部结构与外部结构精确地映射。框架里嵌套框架的想法，可能任何程序员都会认出来这是树的层次结构。Framework只是一个树结构的数据结构，每个框架都依次指向它的子框架。这是一个非常简单的架构。

采访者：你认为集成化软件程序的一般概念是什么？

卡尔：这是一个非常棒的话题。可以在很多层次上谈论集成软件。集成化的一个方面，也是大多数人所想到的，是一个多功能的软件包，它的功能等效于几个独立的软件组合，如电子表格、图形处理和字处理等。这种做法只是将几个功能放到一个集成的软件包中，与微软的Windows系统那样的集成环境相比，不过是同时提供了几个功能而已。人们需要的不只这些。你会希望在不同的功能中能够共享数据，希望能方便地交替使用不同的应用程序。这就是所谓的知识迁移，或命令的一致性。



至少在今天，一个集成化软件包比一个集成化的环境有着更令人信服的优势。在易用易学方面，集成软件包的用户界面源于将用户界面分成两个层面的特征，是有优势的。

用户界面中的两个层面是句法和语义。每个用户界面都有自己的句法，就是那些支配如何与命令本身进行互动的规则或方面——如何呈现命令，如何发出命令，命令的响应或操作是什么样的。在只读存储器（ROM）中配备了工具箱的Macintosh计算机，微软的Windows，还有Gem，所有这些产品在处理命令句法的一致性上都非常出色。通常情况下，所有命令都是以相似的方式发出的，用同样的按键方式或鼠标点击方式发出命令。在不同的功能中，语法是非常一致的。

但还有一个层面对于易用性同样重要，那就是语义层，它涉及指令的含义，或者说应当如何处理信息。大多数环境都不处理语义一致性的问题。今天在Macintosh上有很多数据库，其中很多数据库产品的工作方式都与其他数据库产品的工作方式有很大差异，没有语义的一致性。一个数据库产品会在光标前插入记录，另一个数据库产品会在光标后添加记录。有的数据库产品甚至不使用鼠标，而是用菜单命令。语义差别很大，这样一来，即便通常在高层有着语法的一致性，当你从一个数据库产品转换到另一个数据库产品

时，仍然需要一个低层的知识迁移。

采访者：有什么办法能够让语义在多个功能上保持一致？

卡尔：我强烈地认为，现今还没有技术上的解决方案。在语义一致性的问题上，没有一套编译器或结构良好的面向对象的接口，没有库模块，甚至没有写在纸上的规则。能做到语义一致性的唯一方法就是通过我所说的解决方案小组——由人组成的小组，包含了优秀的设计师和程序员，非常密切地在一起合作。

我们就是这样开发Framework软件的。该产品已被评为头号易学易用的软件。那是因为我们在一起工作时是作为一个单元的，就好像是同一个头脑。

将来我们可能会看到一种我称为“组件软件”的趋势。未来的操作系统有可能会支持更多的模块化软件，可以把不同的模块链接在一起。我们也许能看到不同的模板放在独立的软件包中出售，用户可以分开购买，而不同的模块的语义是高度一致的。但至少在几年内，这些模块仍然必须来自同一个组织。



还需要一段时间人们才能买到一个集成化的环境，然后从一个供应商购买电子表格，从另一家供应商购买字处理器，再从第三家购买数据库，并能在它们之间得到有意义的知识迁移。

采访者：为什么不去标准化呢？那不就可以解决语义不一致的问题了吗？

卡尔：首先，没有任何单一的功能有最优解决方案，无论是电子报表还是数据库。在试图为跨数据库、电子表格、字处理、图形的命令和操作提出一个一致的设计时，事情变得更加复杂了。每个领域都有独特的需求和重要的地方。你需要拿出一个对命令设计和用户界面设计都是最佳的解决方案。

数据集成是集成化软件的另一个领域，可以解决环境所不能解决的问题。有两方面的数据集成。首先是剪切和粘贴，或者说把信息从一个模块传递到另一个模块中。我认为环境在剪切和粘贴工作的方面做得不错。它们通常会说明如何做到这一点，如果在编写模块时已经考虑了剪切和粘贴功能，就可以把电子表格中的数字放到字处理器或图形到去。集成化软件已经做到了这一点。

数据集成的另一个重要方面是让不同类型的数据以它们自己的格式存放在一起。在单个文档中，是否可以实时放入一个电子表格，在电子表格的两边是否可以实时做字处理？是否可以实时放入图形？作为人，我们处理的

是逻辑实体，不关心报表是否混有数字、图形和文字。对我们来说报表就是一个单一的逻辑实体——对细分市场盈利情况的报告需要由不同类型的数据来准确地表达信息。

为什么要求用户从电子表格中分出两个不同的文件来，从图形程序中分出三个文件来，从字处理程序中分出两个文件来？为什么不能只是一个名为“市场营销报告”的文件？这就是实时的数据集成，这可能也是Framework软件的基本驱动点。我觉得，用户需要能够把他们的信息打散成独立的部分，而这些独立的部分必须要有不同类型的数据。在Framework软件中，就让程序去考虑如何来解决吧，用户只需把报表当作一个单一的逻辑实体来处理。

采访者： 计算机的未来会怎么样？在5年或10年后会走向哪里？你认为那时我们还会使用同样的工具和语言工作吗？

卡尔： 我们今天所知道的生产力工具还会使用很长一段时间，它们很好用，这是它们得以存在的原因。所有的系统在建立时仍会是从头开始。未来的挑战之一，应该是拿出一个更好的方法来组合和匹配模块，建立一套灵活的构件基础库了。操作系统环境正朝着这个方向迈进。在每个产品中，不用再对很多语义用户界面重新实现了。因为这个原因，我认为现有的操作系统还会接着用下去。



我希望我们能走向组件软件。有些公司知道如何更好地实现浮点运算或是更好地设计字处理器，用户能够用这些插件模块替换自己程序的一部分。这将是未来10年的趋势。但它是一个艰难的目标，因为我们所涉及的是软件各个独立模块之间的接口，而这是软件设计中我们了解最少的领域之一。我认为很多人并不擅长设计和实现良好、简洁的界面。

我还期待着几年后的某一天，我们已经不再被英特尔8086芯片的分段式结构所折磨。也许它可能是80386。我认为8086有它的优点，但它还是会让我觉得非常苦恼。在开发Framework软件时，它让我们的开发时间和成本结结实实地增加了30%，仅仅是因为和我们较劲的是它将内存截成一些片断的分段方式，而不是保持为一个长长的、连续的内存流。

采访者： 你觉得自己会一直编程吗？

卡尔： 不会，我希望我能有重大的改变，去做些别的事情。编程会让人上瘾。把Framework软件推向市场给我带来的一个大问题是我第一次成了工作狂。在开发Framework软件时，我知道自己是个工作狂，因为我想成为工作狂。

我努力实现给自己设定的目标，成为一个工作狂、耗费我的生活和人际关系是到达目标的一部分。但我可不想永远是个工作狂。

自从完成了原始的Framework软件后，我已经能够回到原来的生活方式了，但我希望在编程或其他一些值得努力的事上能再次消耗我的精力。我在编程这个领域还能再干上几年——我已经29岁了，再干5到8年，那时应该是要做出重大改变的时候了。我的技能似乎更着重于编程以外的事上。我有相当好的口才和文笔，我比其他很多程序员更擅长与人沟通。这不仅对管理很有用，对销售也是很有价值的。

自从有了Framework的创新想法后，我从第一天开始就不得不推销、推销、推销。不是大声叫卖，而是必须一遍又一遍地、清楚简要地介绍那个构想。我演示程序给人看，并要跟每个人解释为什么这个软件是与众不同的，为什么比别的优秀。我们的Framework软件能成功的原因之一就是因为，我和公司的共同创始人及合伙人马蒂・马兹内（Marty Mazner）——他更精通市场营销——能够将底层的技术创新与令人信服的概念演示结合起来。



续写传奇人生

1987年，卡尔与他人合伙成立GO公司，致力于开发带有手写界面的便携式PC。卡尔领导了所有的软件开发工作，其中就有划时代的PenPoint操作系统。Go遭遇了来自微软和苹果的激烈竞争。1991年，Go将硬件部门拆分出来，单独命名为EO。1993年，AT&T收购EO。1994年1月，因为现金危机，GO自身也被卖给了AT&T。GO的PenPoint系统可以在EO手机和IBM电脑上运行，但是消费者对平板电脑的反响并不热烈。AT&T收购GO公司两个星期之后，即决定不再继续投入EO或GO。于是，在烧光风投的7500万投资后，GO公司于当年7月黯然退出历史舞台。

卡尔回顾GO时，曾如此描述GO PenPoint操作系统："可以手持；当时硬件水平允许的轻薄LCD屏幕；从口袋到速记本般大小的尺寸；没有物理键盘，多种输入方法；灵敏的触摸式屏幕键盘，包括点击、滚动、多点触摸手势等一整套屏幕手势，而且无论操作系统还是第三方应用都支持这些手势；即时开启的硬件；整合网络通讯；可选手机通讯模块；第三方程序开放市场。没错，我说的是1991～1993年间的GO，不是今天的iPhone或iPad！"

之后卡尔加入Autodesk公司。作为负责AutoCAD市场的副总裁，他为

AutoCAD的流行立下了汗马功劳。

1997年，卡尔与另外4人联合创办Sofinnova Ventures，并出任总经理，这家高科技风险投资公司管理着5.5亿美元的资金。

2003年，卡尔与自家兄弟一起创立Gee Whiz Labs公司，并担任CEO。他们的主要产品即KeepandShare.com网站，截至2011年夏天，这家网站已有170多万用户。



SYMMETRICAL AIRFOIL GENERATION PROGRAM. JEF RASKIN, 1984

While this program is nothing that experienced programmers will learn much about coding from, it demonstrates the power of being able to embed executeable code into the text produced by a word processor. What this does is to make the mechanics of documentation simple. Using a typical program editor for extended text is usually bothersome enough a chore so that our program comments become terse and more difficult to read. I have observed that when I document a program in this thorough fashion, it often runs on the second or third try, whereas when I do not document carefully, much more debugging is required.

Please note that this text you are reading is part of the program (or, equivalently, that the program is part of the text).

The program itself plots, on an Apple II, given Information Appliance's SwyftWare environment, a family of airfoils that I am developing for my aerobatic model airplanes. It does it by a transformation of a circle, much as the Joukowsky transformation does. This permits relatively easy calculation of the lifting characteristics of the airfoil, although a program to do that is not shown here.

The program begins by setting up graphics mode and a few constants. First, for graphics mode:

```
90 HGR
```

The constants include

```
100 pi = 3.1415926
```

and one that determines how much the airfoil will "cusp" or be concave toward the rear.

```
110 c = .81
```

Another important variable controls the thickness ratio of the airfoil.

```
120 f = .22
```

The program works by using the usual pair of formulae

```
x = r cos t        y = r sin t
```

to generate a circle parametrically by letting t vary from 0 to 2 * pi in convenient steps. r is chosen to be .5 so that a circle of diameter 1 results.

```
130 FOR t = 0 TO 2 * pi STEP .04
140 x = .5 * cos(t)
150 y = .5 * sin(t)
```

The circle is centered on the origin, and we first move it to the right by .5 so that all the x-values are positive, and are, in fact, in the range (0,1).

```
160 x = x + .5
```

To understand how this transforms a circle into an airfoil rounded at one end and pointed at the other, note that if we multiply each y-value by the corresponding x-value we will, near x=0, have very small y-values, yielding a point. But as we approach x=1 the curve approaches a circle.

```
200 y = y * x^c
```

The exponent of x changes x's effect. Many functions of x will do for this role.

Airfoils tend to be long and thin. Thus we want to squash the shape in the y-direction. The thickness ratio is thus applied to y. This could have been done in the previous line of code, but it is separated out for clarity.

```
210 y = y * f
```

We now have -f < y < f and 0 <= x <=1. This has to be translated to Apple screen coordinates where 0 <= x < 280. Again, these transformations could have been included in some of the earlier program statements, but for clarity the plotting is separated from the generation of the foils. The 250 is chosen as conveniently large for the screen. This will make x vary from 0 to 250

```
400 x = x * 250
```

and to keep things square, we apply the same scaling to y, remembering that y has already been decreased by the f factor. We also have to make y always positive. Obviously, we will go off-screen if f is too large, but we will assume smart users who read this documentation and behave accordingly rather than bother with too much error checking.

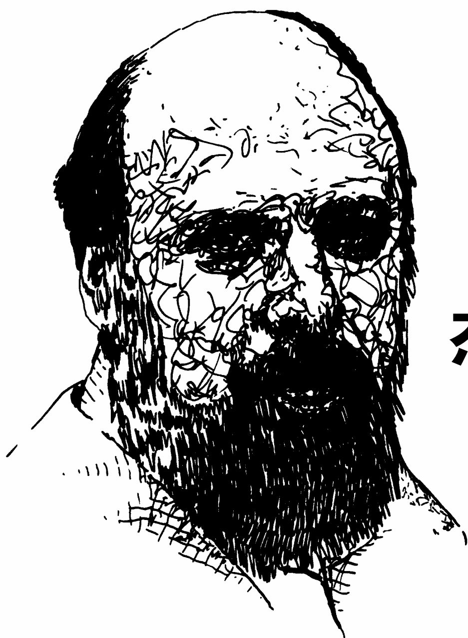
```
410 y = y * 250 + 100
```

Now to plot these points, and close the loop, and then go into an infinite loop so that the computer will not mess up the screen with messages.

```
500 HPLOT x,y
510 NEXT t
520 GOTO 520
```

All that's left is to type and use the CALC command on "run". You will have to modify the program to run on systems other than SwyftWare-equipped Apple II's.

这个程序演示了拉斯金如何将可执行代码嵌入到字处理器产生的文本中。



14

杰夫·拉斯金



苹果电脑公司Macintosh项目发起人杰夫·拉斯金（Jef Raskin）多才多艺。他担任过旧金山室内乐团的指挥，持有包装设计、飞行器结构和电子学领域的多项专利。他还是个艺术家，作品在纽约现代艺术博物馆和洛杉矶美术馆展出过。他目前担任信息设备公司（Information Appliance Inc）的CEO。1943年，拉斯金出生在纽约市，用他的话来说，差不多跟数字计算机同龄。

在纽约州立大学石溪分校，他主攻数学、物理、哲学和音乐，在校期间，他多次赢得奖学金，并获美国国家自然科学基金会（National Science Foundation）的资助。经过5年学习，他大学毕业并获得哲学学士学位。之后他又在宾夕法尼亚州立大学取得计算机科学硕士学位，后来成为加州大学圣迭戈分校（UCSD）视觉艺术教授。他在那里执教5年，同时还担任第三学院[①]计算机中心主任。

他后来毅然决然地从大学辞职了——关于为什么要离职，他并不想多说。随后，他成为一名职业音乐家，从事教学和指挥。当8080微处理器面世时，拉斯金创办了Bannister & Crun公司来利用这项新技术。公司找到一块有利可图的细分市场，专门给Heath、苹果、美国国家半导体和其他公司编写手册和软件。

① Third College，参见http://en.wikipedia.org/wiki/ Thurgood_Marshall_College。

1978年，他加入苹果电脑公司，成为第31号员工，担任出版经理。后来他又担任高级系统的经理，组建了创造Macintosh的团队。1982年，他离开苹果公司，到丹麦数据处理学院（Dansk Datamatik Institute）任教，之后回到硅谷创办信息设备公司。

提到本书书名时，杰夫·拉斯金马上指出，他本身并不是程序员，而是一名设计师或"元程序员"（metaprogrammer），而且他的公司也跟传统软件无关。拉斯金处处显示着自己的与众不同。他是个发明家，对于如何利用新软件和创新方法让电脑运转起来，有着独到的想法。他不是坐在电脑前写源代码的那种人。随着软件越变越复杂，越来越多的人接受将规格说明书转换成代码的技术培训，程序员和软件或系统设计师之间的这种差别也会越来越被人了解。

拉斯金新开公司的办公室坐落于帕洛阿尔托的大学街（University Avenue），这片城区似乎成了众多苹果电脑前雇员的聚集地。信息设备公司的办公室不像其他许多软件公司那样雅致奢华，实际上他们的办公室面积不大，有点昏暗，总之非常普通。它们原本也许是小电器公司的办公室，我猜这也是拉斯金微电脑理念的一部分。



拉斯金胡须灰白，些微秃顶，眼神警觉而锐利，一副经验丰富的教授模样。他招呼我时有些上气不接下气，他解释骑车上班的路上出了点事故，结果来晚了。这让他看上去有点心绪不宁。他领我进了他的小办公室，并稍稍整理了一下东西。他办公桌后面一侧放着一台Apple IIe电脑。靠另一面墙放着雅马哈键盘合成器。我身后是一辆10速自行车，一个自行车气泵，还有一个像是东方神龛的小东西。

拉斯金生气勃勃，精神焕发，兴奋溢于言表。有时候，我感觉自己仿佛置身游乐园。某一刻，他会给我做过目难忘的新产品演示，犹如变魔术一般；过一会儿，他又会耍起小把戏；接着，他会转过身，在合成器上弹上一曲；然后，他又掏出一辆小模型车，在桌子上来回把玩，一边谈论着微电脑的过去、现在和未来。

在他眼里，微电脑不是什么耀眼的、神秘的、难以捉摸的机器，而应该是一台不显眼的、易用又实用的电器。

*　　*　　*

采访者：你最出名的就是在苹果公司创造了Macintosh。你在其中扮演了什么角色？

拉斯金：1979年，苹果公司正致力于开发Lisa。不管你信不信，它最初的目标是开发一款字符界面的机器。我在苹果公司担任高级系统经理，对Lisa很不满。它非常昂贵，我认为苹果公司不够理智，居然以小型机的价格去跟DEC、通用数据公司和IBM比拼。

70年代初，我在斯坦福大学人工智能实验室做访问学者期间，在施乐PARC度过了大量时光。在我看来，施乐PARC在位图屏幕、通用键盘和图形方面的研究真是不可思议。因此，我极力游说，说服苹果公司把Lisa改成位图机器。我把施乐和苹果公司撮合到一起。施乐一度持有苹果公司10%的股票。

我提议开发一款容易使用并融合文字和图形的电脑，售价在1000美元左右。乔布斯认为这想法太疯狂了，这种机器不会有销路，我们绝不要这种玩意。他试图否决这个项目。

因此，我越过乔布斯直接找到当时的董事长迈克·马库拉（Mike Markkula），和他探讨我的想法。幸运的是，马库拉和时任总裁迈克·斯科特（Mike Scott）叮嘱乔布斯别管我。

我招揽了早期成员：巴德·特里伯（Bud Tribble）、布里安·霍华德（Brian Howard）和布雷尔·史密斯（Burrell Smith）。我们搬到另一栋大楼，打造Macintosh及其软件的原型，搭建并让它运行起来。后来，乔布斯接手后，他还编了个故事，把Mac项目称为“海盗行动”。我们并未像他后来说的那样，试图让这个项目远离苹果公司。我们跟苹果公司其他部门的联系十分紧密，我们只是尽量不让乔布斯干预这个项目。在头两年，乔布斯总想着要毙掉这个项目，因为他不明白它到底是怎么回事。

最初的Macintosh设计得既精心又合理。最后，苹果公司所有人都意识到，这是公司继Apple II后推出新产品的希望所在。随后，乔布斯接管了这个项目。他径直走进来，对我说：“我接手Macintosh硬件，你可以负责软件和出版物。”他抛出软件设计，要求Macintosh软件与Lisa保持兼容，并坚持使用鼠标。机器变得更大、更复杂，也昂贵得多。现在它跑起来黏糊糊的。你有没有用过MacWrite？我们这里把它叫做MacWait。又过了几个月，乔布斯对我说：“我来接手软件，你可以负责出版物。”于是我回答：“你也可以接手出版物。”说完便离开了。那是1982年5月。他和马库拉对我说：“请别



走。再过一个月，我们会提供一份你无法拒绝的薪酬。”于是我给了他们一个月时间，他们开了一份薪酬，我没接受。

采访者：不过这款电脑成功了。乔布斯肯定多少有些贡献。

拉斯金：他的确做了一些很重要的工作，尤其是在苹果公司初期，使得苹果不只是一家小型电脑公司，而是成为一家举足轻重的公司。他也很有想法：把Apple II装在漂亮的一体式机箱里，做市场推广，四处招揽优秀人才，确保它有很好的手册。

采访者：你好像非常在意自己工作的功劳归属问题。

拉斯金：我并不想独揽Macintosh的所有功劳，它是团队努力的成果。就算乔布斯只是因他真正为业界所做的贡献而得到好评，那也已经够多的了。但他还硬要把别人的功劳据为己有，我认为这不合适。

近期《新闻周刊》上的一篇文章把我逗乐了，他说：“我还有几个不错的设计。”他从来没做过什么设计，他一款产品都没设计过。沃兹（Steve Wozniak）设计了Apple II。肯·罗斯穆勒（Ken Rothmuller）等人设计了Lisa。我和我的团队设计了Macintosh。温德尔·桑德斯（Wendell Sanders）设计了Apple III。乔布斯设计了什么？什么也没有。



采访者：你觉得自己在苹果公司的经历很糟糕吗？

拉斯金：不。最初几年，76年到80年，非常美妙。我度过了一段美好时光，了无遗憾。早期跟乔布斯和沃兹尼亚克共事很快乐。但是从1980年底到1982年，在苹果公司工作却犹如噩梦一般。

采访者：离开苹果公司时，你认为自己还会回到这个行业吗？

拉斯金：我觉得自己再也不会做这行，或者别的什么蠢事，比如再回到硅谷工作。我厌倦了一周工作7天。

离开苹果后，我到丹麦重拾教鞭。那时我刚结婚，还在度蜜月，期间有了这个想法。我意识到自己这些年的努力全都白费了。我总是千方百计地想设计一台更好的电脑，实际上我要的根本就不是电脑。我想要的是像电器那样工作的东西。我觉得这想法太妙了，不分享不足为快。

采访者：就是这个想法促使你创办了信息设备公司（Information Appliance）。你怎么会用“设备”这个词来反映你在这里开发的产品？

拉斯金：不知你发现没有，并不存在所谓的美泰克[1]用户组。操作洗衣机不需要求助互助小组。你只消把衣服扔进洗衣机，按下按钮，衣服就洗干净了。要做信息处理，我想要的并不是硬件和软件；我真正想要的是一台能完成任务的设备。我会做什么任务呢？调查表明，85%的个人电脑用户使用字处理，所以我需要的是字处理器，一个超棒的、世上最好的字处理器。不过我记性不是太好，只记得住10到15个命令，这就是为什么我使用的系统只有5个命令的原因。这样我凌晨3点醒来，一起床，就可以走到电脑前，直接记下自己的奇思妙想。

采访者：你的意思是你已经尽量简化了系统？

拉斯金：没错。我们设想一下，假如烤面包机是电脑公司设计的，它会是什么样子。你一早醒来，早餐想吃一片吐司。你会先打开烤面包机的开关。如果烤面包机是通用电气设计的，你会把吐司放进去，然后转身走开，不过，这台烤面包机是电脑公司设计的，一切就大不一样。那么，到底会发生什么？首先，它会执行两分钟的烤面包机自检。然后你将系统磁盘放进烤面包机，并启动系统。之后，你插入早餐磁盘，并键入“Load TOASTED.CODE”（装载烤面包代码）。



那么，接下来又会发生什么？接着烤面包机上会出现菜单。它会提示：“你想要什么样的面包？”如果它是个加州程序，就会显示：羊角面包、百吉饼、英式松饼、全麦，白面包当然在最底下。

它们会被标记成A、B、C、D和E，早上想吃松饼的话，就按下C。结果什么反应都没有，因为你忘了敲回车。你以为机器很智能，会直接回应你按下的C，但令人无奈的是，你还都得敲回车。

你认为现在该有结果了吧？

采访者：嗯……

拉斯金：当然不是。它可是由电脑公司设计的。它会显示：“你确定吗？”现在你只想把它扔出窗外。你怒了没有？难道面对电脑这么多年你还没出离愤怒吗？因为你为它掏了几千美元，还得忍受这愚蠢的一切，世界上其他地方也不例外。成千上万的人每次使用电脑时都得忍受这种荒谬的过程。

于是，你输入“是”并敲回车，却收到一条错误消息，因为你还应该敲

① Maytag，美国家用及商用电器企业，创办于1893年，2006年被惠而浦收购。

另一个键。你赶紧查阅手册，但一无所获，因为这手册描述的原型系统已不同往日。最后，你把面包放进第二个面包槽，提示你想要略微焦的、半焦的还是烤焦的面包，电脑会问："是否需要保存这份早餐配置，免得下次还要重选？"于是你输入"是"，它提示你将磁盘放入第一个槽，但是你突然发现手头没有格式化过的磁盘。

你打电话给经销商，询问有什么办法可以在格式化磁盘的同时保存这些信息，而且还不会弄丢早上做的这一切。经销商回答："可以啊，只需掏3000美元买这台装有MS-DOS 9.8的硬盘系统，它会帮你搞定所有问题。它配有一本手册和一个手推车。"手推车是搬手册用的。这么一来你上班就要迟到了……但是现实就是如此，我们的产品很能说明这种困境。不妨到我的电脑前看看。

[我们在一台Apple IIe前坐下，这台电脑配有SwyftCard和键盘标签。]

瞧这个。驱动器里没有磁盘，我想输入一条信息："记得带些牛奶回家。"你觉得怎么样？我打开电脑，开始输入。不需要输入命令，不用插入，也不用找编辑器，我可以直接开始输入。



现在我想打印这条信息，放口袋里以备不时之需。我只消按一个键，它就会打印出来。这不是很方便吗？

采访者：*用这个可以烤面包吗？*

拉斯金：不行，面包屑会掉进磁盘驱动器。

采访者：*用户用这台设备还能做什么？*

拉斯金：我们可以轻松完成计算。以前，每当我使用字处理器并想做些计算的时候，我就得掏出袖珍计算器，不得不使用一个单独的计算程序，或打开SideKick[①]；在Mac电脑上，你会调出计算器，把计算结果粘贴到文档里。我们还支持远程通信功能。

采访者：*这些功能全在同一个程序里？*

拉斯金：当然。所有应用程序之间都没什么不同。什么是字处理器？你用它来产生文本、四处移动、碰到出错的予以修改，以及查找内容。什么是

① Borland公司于1983年推出的个人信息管理程序，包含计算器、记事本、行事日历等，详见：http://en.wikipedia.org/wiki/SideKick#Mac_version。

远程通信程序包？你用它来生成文本，或接收别人生成的文本。它不是从键盘输入或从打印机输出，而是在电话线上进进出出。什么是计算器？你用它生成数字，那也只是文本，计算结果应该回填到文本中。有一天，我恍然大悟，如果这些应用程序都做同样的事情，何不写个小程序把这些事情一并做好呢？

采访者：嗯，你开发的产品涵盖了所有这些功能，这个产品是什么样的？

拉斯金：[拉斯金拿起一块简单的插卡。]它叫SwyftCard。这是Apple IIe上用的插卡。并不怎么复杂，对吧？这就是为什么我们能把它卖到89.95美元的原因。它是如此友好，如此简单，其实有顾客告诉我们，看到这个之后，他们卖掉了自己的IBM个人电脑或Macintosh，就为了买台Apple IIe装上SwyftCard。

采访者：这个插卡只能用在Apple IIe上吗？

拉斯金：SwyftCard目前只支持Apple IIe。我们很快会针对Apple IIc开发一款叫SwyftDisk的产品，提供相同的软件功能。SwyftDisk可能会在3月初推出。



采访者：你计划在其他电脑上实现这款产品吗？

拉斯金：对于还没做出来的东西，我们从不宣扬，这是我们公司的行事风格。在得知这款产品的第一时间，你就可以下订单。另外，我再给你展示一下这个系统的其他一些优点…… [拉斯金回到电脑屏幕前。]假定你忘了保存部分文本，直接载入另一个文件，通常那些文本就会消失不见。这个系统非常智能，绝不会丢失任何东西。不过我还会做得更过分。[他从电脑前起身，将一张磁盘放在墙上挂的一块大磁铁上。]现在这是一张完全未格式化的空白磁盘。我会用这块磁性很强的磁铁来确保它是空白的。我会将磁盘插入磁盘驱动器，并输入一些文字，然后使用DISK命令[此时磁盘吱吱响了4秒钟]。现在，我将做件更糟的事情，直接把机器关掉。不过，我保证机器重新打开时，这些文字已保存妥当。[他重新打开电脑，4秒钟后那些文字又出现在屏幕上。]看见没？这个系统只用短短4秒钟就可以将文字保存在一张完全未格式化的空白磁盘里，而DOS格式化一张磁盘就要用掉近半分钟。不仅如此，我们打开这个文件时，甚至连光标都会恢复到上次关闭时停留的位置。

采访者：不错，它很智能，不过速度快吗？

拉斯金：它能处理的输入速度远快于人们所能达到的输入速度，在你输入文字时，它会自动处理文字换行、格式和分页等。光标移动手法要比鼠标快。它不仅本身就快，而且你用起来会更快，因为你不受羁绊。

采访者：这么快的速度，你是怎么实现的？

拉斯金：非常非常简洁的设计。这个系统不抽烟也不喝酒。这体现了我这阵子开发东西的方式。

采访者：大多数用户都被鼠标搞晕了，不过它有一定的市场接受度。为什么你不喜欢它呢？

拉斯金：我讨厌鼠标。鼠标需要你移动手臂，会减慢速度。我不想在Macintosh上使用鼠标，但是乔布斯坚持要用。当时，一切都是他说了算，不管想法好坏与否。

采访者：你的整个做法似乎逆行业发展趋势而行，这行的趋势是制造更大的计算机以容纳更大的程序……



拉斯金：是的。我们不追求大而又大，我们奉行好上加好。在向投资人描述这个项目时，我告诉他们："我们将打造一个字处理器，支持信息检索和远程通信包，只有15个命令和64KB的代码。"其他所有公司给出的都是数以百计的命令和几百KB的代码。我们公司（信息设备）惊讶地发现它最终只需5个命令。随着时间的推移，项目越来越简单而不是越来越大或越来越复杂，我见过的只此一例。

整个硅谷的人都在比较UNIX和MS-DOS。你需要拿它们做什么呢？把它们扔一边儿去，不必理睬那些鬼话。也许计算机科学家需要它，但对于只想把事情做成的普通人，你并不需要。你需要操作系统吗？不。我们要抛开这个观念。诸如VisiOn[①]、Gem[②]和Windows等应用程序只是对隐藏其下的操作系统的装饰，而我们这个程序底下没有操作系统。当你给某样东西抹上浓妆后，你知道会发生什么？你看到的就是浓妆艳抹的样子。

因此，这个程序运行在老旧的Apple IIe上，处理器只有1兆赫兹，但从用户的角度来看，它的运行速度比IBM、Macintosh、大型机、SuperVax或其他机器都要快。

---

① http://en.wikipedia.org/wiki/Visi_On。

② http://en.wikipedia.org/wiki/Graphical_Environment_Manager。

采访者：这个项目的动力与驱使你设计Macintosh的动力有所不同吗？

拉斯金：部分动力相同，部分不大相同。就Mac电脑来说，我试图尽我所能创造最好的电脑。而创办信息设备公司时，我不再试图制造电脑。我只想让所有人都能从计算机技术中轻松获益。

这个想法可以追溯到60年代，那时我还在加州大学圣迭戈分校担任跨学科教授，并在视觉艺术系任教。我不仅是计算机科学家，还是艺术家和音乐家，我喜欢给艺术和人文学科的人教授计算机编程。我还教计算机编程、计算机艺术和计算机电影制作，对象往往是那些不从事这些领域的人：女修道院院长班，或者中小学三四年级的学生。

采访者：进入苹果公司前，你的经历以学术为主。你加入苹果公司之前具体从事哪些工作？

拉斯金：哦，我六年级时的项目是用继电器和大开关搭建数字计算机。我在纽约州立大学石溪分校学习数学、哲学、音乐和物理学。在宾夕法尼亚州立大学，我一开始攻读哲学方面的博士学位，但最后取得了计算机科学硕士学位。在加州大学圣迭戈分校，我开始念的是音乐博士学位，最后却变成了一名艺术教授。我当了5年教授，然后进入斯坦福大学人工智能实验室做访问学者。

接着，我到旧金山室内乐团担任指挥，并在当地教授音乐。然后，个人电脑问世了，我想："嘿，电脑又勾起了我的兴趣。"我买了一套Altair开发套件，并把它组装好。我发现那些说明书糟糕透顶，于是成立了一家文档编制公司Bannister & Crun，该名字取自一个英语电台老节目Goon Show[①]中的两个人物。我主要给Heath、国家半导体和苹果公司编制手册。另外我还开了一家模型飞机公司。

采访者：你觉得音乐、艺术和计算机之间有共同之处吗？

拉斯金：不能相提并论。但我确实一直努力做那些能让人们快乐的事情。我喜欢当音乐家的一个原因是，世界上的音乐家很少干什么坏事。如果你是物理学家，可能发明会爆炸的东西。但音乐家绝不可能干这事。艺术家可以创作政治宣传海报，但对音乐家来说，所创作的通常是中立的东西。我一直热衷于做那些会让人们快乐的事情。



---

① 详见http://en.wikipedia.org/wiki/The_Goon_Show。

使用SwyftCard的人不会弄丢文件，花费在工作上的时间会少很多，也不会那么焦虑。相比其他产品，他们使用SwyftCard时更加快乐。我力求做出对这个世界真正有用的东西。还有一点，这个系统也适用于盲人，这要归功于光标的移动方式。对于盲人而言，像Macintosh这种以视觉为导向的系统根本用不了。

目前，我们的系统已经在退役军人管理局（Veteran's Administration）、西部盲人康复中心（Western Rehabilitation Center）和感官辅具基金会（Sensory Aids Foundation）等机构投入使用，他们向我们反馈说："嘿，盲人也能用这个产品。"因此我们现在做的产品将惠及大批人群。

采访者：你还有时间追求艺术吗？

拉斯金：我不做视觉艺术，但我还是个音乐家，尽管演奏的机会并不多。现在经营公司，我没有充足的时间来练习。偶尔，我会受邀在朋友的婚礼上演奏。我也是靠到酒吧和夜总会演奏才读完研究生的。在宾夕法尼亚州立大学，每周三晚上我会为老电影做钢琴伴奏。我家里有一架9英尺长（约2.75米）的三角钢琴。我喜欢弹奏巴赫和莫扎特。



采访者：这么深厚的学术背景想必让你在苹果公司成了热门人物吧？

拉斯金：在苹果的最初几年，全公司只有我一个人拥有计算机科学学位。他们要是知道我其实当过教授和计算机中心主任，有可能不会让我进公司，所以这些我并没跟其他人提起过。

采访者：为什么？

拉斯金：因为苹果公司早期对学术研究抱有偏见。我得以进入公司，编制出不错的手册，是因为我对沃兹说："你是硬件方面的专家。"或对兰迪说："你是软件方面的专家。"然后我告诉他们："好吧，我对那些东西不是很在行，我只知道怎么写作，因此我不打扰你们，你们也别打扰我。"

采访者：开发SwyftCard用了多久？

拉斯金：公司成立大概有3年了，不过我们也在忙其他事，因此很难说用了多久。我们本来不打算批量生产Apple IIe专用插卡，但这想法似乎太妙了，要是不发布这款产品，我就觉得浑身不对劲。我在苹果和施乐公司见过大量好产品，但都仅止于公司内部，从来没有上市过。

采访者：找到创办信息设备公司的支持是不是很难，还是好想法带来的力量以及从Macintosh上获得的声望帮你渡过了这一关？

拉斯金：在制定好SwyftCard的功能细节后，我要求苹果公司和施乐PARC的几个朋友保证不泄密，然后把技术规格拿给他们看。他们看了一眼说："不可能，这行不通。我们已经为之努力了几十年都不行。"然后，第二天他们都给我打电话说："嘿，你知道，它看起来好像行得通。"于是，我决定自掏腰包招揽几个程序员，看看自己的想法能否实现。我决定用FORTH实现SwyftCard，因为FORTH语言相当紧凑，实现起来也不费钱。这不是我最喜欢的语言，但我认为它很适合这个特定的应用。我一贯认为要干好活得选对工具。

我出去雇了一个FORTH程序员和另外几个人，多数是我自己的朋友。没人加入过苹果公司。我没去沾惹这家公司。我不想惹上什么法律麻烦，而且当时苹果很难搞，我也担心发生这种事。

采访者：你有没有开展密集的营销活动以推广你们的产品？

拉斯金：我们在市场营销上的预算不多。只是花了点钱，做了几个广告，把这个信息传递给那些想要这款产品的人。市场反应不错，都说这是很棒的产品；销售额也节节攀升，增长速度远超我们的预期。

采访者：你会提防过度的市场营销和炒作吗？你是否认为这已经给这个行业造成了问题？

拉斯金：当然是。我在杂志上看到过一整版全彩广告，兜售Apple IIc用的液晶屏，于是我打电话给这家公司。他们告诉我说："我们预计该产品会在10个月后上市。"一整版彩色广告！在我看来，他们完全是在浪费钱财。

采访者：作为学校教授、杂志记者、手册作者和电脑设计师，你一直在传递这样的信息，即电脑的好处理应惠及普通大众。信息设备公司是否只是这种信息的另一个载体？

拉斯金：这家公司如果能取得成功，我将再次影响到数百万人。把这信息传播开来的媒介就是产品。如果公司赚得盆满钵满，大家就会听到我的信息。因此，很遗憾，当下传递信息的方法只能靠赚钱。

如果我想告诉全世界位图屏幕太棒了，不管我写多少文章，没人会理会。施乐公司发表过几十篇文章，有谁在意？但是，把Macintosh做出来并以公



道的价格卖上几万台，你猜怎么着？那些买过一台或有所耳闻的人都会发现位图屏幕和图形的整个想法，还会发现单独的图形和文字模式已不再需要，字母不过是图形的另一种形式，不需要额外的硬件就可以制作花哨的字体。

这样的状况无疑令人生厌，但是很遗憾，这就是当下的现实：你赚的钱越多，愿意聆听你的人就越多。如果你的话没被《财富》、《福布斯》或《华尔街日报》引用过，就没有人会理睬。如果你说某样东西能赚很多钱，无论你说的是真是假，人们都会洗耳恭听。

采访者：你在大公司和小公司都待过，你怎么看待两者的异同？

拉斯金：你应该听过苹果公司规模扩大时乔布斯说过他打算怎么做。他这么快就忘了自己的理想。

相比之下，沃兹就保持得挺好。在苹果公司，挣到不少钱又没什么改变的人屈指可数，布里安·霍华德便是其中一个。他这人很棒。我不清楚他身家有多少，不过他仍然开着10年前我以75美元卖给他的那辆车。



采访者：你认为你的企业面临的最大问题是什么？

拉斯金：到底该怎么推销与众不同的东西？这是最大的问题。外界还没有准备好相信有这东西。这就像在苹果公司成立初期，他们问："这有什么用？"我们无法给出很好的答案，因此他们想当然地认为机器的销路不会太好。但我非常清楚自己打算采用口口相传的方式销售产品。有些人会使用它，然后说："这个产品真不错，把我的活儿搞定了。它没有15种字体。我没法打印出5英尺长的老哥特式横幅，但我确信可以如期完成那篇文章。"这就是我可以用来销售产品的方法。将来，人们会明白的。

这就像我第一次启动Macintosh时，包括乔布斯在内，没人知道图标或位图屏幕是怎么回事。现在，它们已经成了这个行业的常用词汇。我以前做得到，我会再次做到。

采访者：你怎么看待计算机行业当前的不景气？

拉斯金：我认为这就是我们会赚很多钱的原因。在我看来，行情低迷的原因在于打算购买如此难用的机器的那些人都已经买了，现在只能卖给迫不得已用这些机器的人。之所以说迫不得已，是因为他们不得不用它来工作，或者是因为别人告诉他们除此之外别无他法。或者因为同一条街上的邻居家有一台。现在，对于真正易用又好玩的产品来说，市场空间很大。我认为复杂机

器的市场已经饱和，而现在我们公司将开辟一个全新的市场。

在你看过我们的系统后，回去再用你的系统，每次你都会不禁感叹：“要是用信息设备公司的产品，早就完成了。我就用不着坐在这里干等，急得搓手，我也不会错过那些灵光一现的奇思妙想。”我正致力于售卖让人更快乐的系统。使用我们的产品，你可以完成更多工作，烦恼也会更少。

采访者：你认为计算机最终会做字处理、电子表格之外的其他事情吗？你怎么看待人们对于电脑帮你打理家务所做的各种承诺？

拉斯金：我从来不太相信那些，原因我会一一说明。首先，家里会有带微电脑的专用系统，执行各种不同任务。其实目前已经有不少这样的系统，只不过你没把它们看作电脑：它们都是电器。当你使用高档微波炉时，你有没有想到它里头实际上装了微电脑，有小容量的RAM和ROM，并且还会运行程序？谁会去想那些呢？你只需设定时间，把食物放进去，热一热当早饭。那里头隐藏着一个微处理器。我们有很多隐藏的微处理器。但是电脑会做那些吗？不可能。首先，你可曾见过哪两种电脑产品能真正兼容的？你认为制造电动窗的XYZ公司会让他们的产品与G公司的电脑兼容吗？你知道，在你的电脑上尝试运行其他厂商的打印机，结果有多么糟。这行不通，以后也一样。

采访者：除了电脑方面的工作，你还做其他具有创造性的的项目吗？

拉斯金：五花八门。我是个典型的发明家。我做过大量不同领域的项目。例如，我有一家公司制作无线电遥控模型飞机。另外我正在做一个钢琴方面的新设计。可能它的成功不会超过过去百年来钢琴的其他创新，也就是说，它可能会彻头彻尾地失败。我的新钢琴声音很美，但我不确定它是否会走向市场。

我正在开发一款数控铣床，售价在5000美元以下，好让那些小工厂也买得起。目前市场上这种铣床价格都在3万到10万美元之间。现在我家里已经有一台能用的数控铣床。

采访者：翻阅你的产品手册，我注意到两点：第一，对于如此注重简单的产品而言，它非常厚；第二，它非常全面和容易阅读。这本手册是否反映了你对电脑文档的体悟？

拉斯金：这本手册的整个出发点就不同。它认为准备使用这本手册的是活生生的人，就像我和你。我们不希望找个答案还得查阅一百本手册，另外我们



也喜欢读英文。因此，在启动这个项目时，我决心写出最简单、最齐整、最出色的手册。

出版这本手册之前，我们不辞辛苦地编写和排版手册初稿，我们知道它不会正式发布，但它看起来真的很不错，因此人们并不会认为他们拿到的是手册初稿。为什么？为了让人们尝试它并从中有所收获。我们希望他们感到自己拿到的是真正已经完成的系统。直到真正做出了产品，我们才开始编制最终的手册。当你运营自己的公司时，你有机会设法把事情做对。在我待过的公司中，我写过的手册被弃之不用，然后重写一遍的状况还从没发生过。我们还做了所有那些傻里傻气且业界没几家公司会做的事，比如把功劳归功给参与过项目的所有人。

手册这么厚是有原因的。我们的想法是：如果拿到这本手册，你身边还有别的设备，那么你用不着查看其他手册就能找到所有问题的答案。我们的手册涵盖了绝大部分答案，不仅针对我们的设备，还针对其他可能涉及的几乎所有东西。我们认为这有别于大部分手册。如果你想知道该产品为什么如此简单，就可以读原理图和“操作的硬件原理”。没什么秘密可言。如果你想知道它的速度为什么这么快，可以阅读“操作的软件理论”。另外还有一篇“操作的用户界面理论”。我在别的地方从没见过。这本手册包含完整的术语表，有助于你了解这个行业。我们用到的每个术语都是明确清晰的。术语表不只有一两页，而是一份真正的术语表。这本手册还包含不同页面之间的交叉参考索引。



采访者：我发现你有点瞧不起那些不使用简单英文命令和文字的用户界面系统和手册，是这样吗？

拉斯金：我经常猜不出那些图标的含义，但我知道英文单词的意思。对我来说，把英文翻译成西班牙文和法文，要比制作世界各地的人们都看得懂的图标更为容易。

我朝这个方向努力了很久。直到离开硅谷，让自己的头脑清醒了6个月，我才突然发现自己一直找错了目标。

采访者：你指的是在硅谷时自己一直身陷复杂性的漩涡之中？

拉斯金：图标、窗口、鼠标、庞大的操作系统、臃肿的程序、集成软件包……我想提醒世人：有两样东西在同一张菜单上，并不代表它们配在一起也好吃。

采访者：这是你向硅谷传递的信息吗？向复杂系统的创造者展示我们还有简

单的替代方案？

拉斯金：是的。

采访者：你觉得这本手册是你当技术作家时的巅峰之作吗？

拉斯金：我是以自由撰稿人的身份进入计算机行业的，曾为*Doctor Dobbs*、*Byte*以及现已停刊的*Silicon Valley Gazette* 等杂志撰稿。

我在苹果公司担任出版经理时，写过几本颇受好评的手册，但是这本手册超越了我以往写过的任何东西。不过这本手册不能算是我写的。主要作者是戴维·阿尔佐封（David Alzofon），我写了一半不到。

整个系统构建在几条规则以及若干使系统易于使用的心理学原理之上。如果读过这本手册的"操作原理"部分，你就会发现我们公司有一套人类行为方式的理论，我们正是借此设计我们的产品。

采访者：那是什么原理？

拉斯金：这比较简单。我会问自己："是什么促使人们形成习惯？"当我使用一个系统时，如果可以全神贯注于自己要做的事情而不是这个系统本身，那我就是最快乐的。系统不应该侵扰用户。我系鞋带完全是出于习惯，并不会真的思前想后一番。我不会把注意力放在系鞋带上。我希望我的系统非常简单，它不会让用户分心，妨碍他们完成主任务；我希望我的系统用起来很自然，就像习惯一样。因此我会问自己："到底是什么促使习惯的形成？"

嗯，我看过的心理学方面的书籍常常会说，当你反复以同一方式做事，而且倾向于以同一方式做事时，也就形成了习惯。从这上面可以立刻受到一些启发。比如你自己开车，假如每到周四，油门和刹车踏板就互换位置，它会让你撞上或撞倒墙壁，这样你肯定受不了。然而，我们的电脑总是通过所谓的"模式"（mode）来改变周遭事物。一个系统应该是"无模式的"（modeless）：同样的用户操作应当始终有同样的效果。

SwyftWare本质上是无模式的。不知什么原因，大多数电脑设计者都会以提供许多种方法完成某件事为乐。如果做某件事有15种方式，他们认为这就是给你自由。实际上，字处理器里的大多数命令，大部分用户都不会用到。其中有几个命令是大家一直会使用的。即使他们已经读过手册，知道使用特定技巧可能会提高效率，他们也不予理会。他们每次都使用同一组命令。

在Macintosh上，所有鼠标命令都有对应的键盘命令，因为没有人愿意一直使用那愚蠢的鼠标。好吧，但是只要提供了一种以上的方式，你在考虑



如何操作时就会有一丝迟疑。即使你已经养成习惯，有时在做某些操作时，你也会想："哎呀，有没有更快的方法？"

因此，不仅一个动作应当只做一件事，而且达成某个目标也应该只有一条路径。你一直采用同一种方式，再也不用思前想后。我们称之为"单一性"。我的系统还没达到完美的单一性，不过已经达到我力所能及的程度。（笑）也许有一天我会知道如何把系统做到完全单一化。

采访者：你认为人工智能程序可以对社会有何贡献？

拉斯金：人工智能教会了我们大量关于我们自身以及知识的内容。非常廉价的机器上用不了靠谱的人工智能程序，至少目前行不通。

真正的人工智能有点像宗教。从前人们都说天空上面就是天堂和天使。然后你搭乘火箭飞船到了那里，发现并不是那么回事。于是你知道原来的理解有误。一旦你实现了某样东西，它就不再是人工智能。



曾几何时，国际象棋博弈程序一度被认为包含人工智能的成分。当我还在念研究生时，只要学会编写国际象棋博弈程序，你就可以拿到人工智能博士学位。现在你花上29.95美元就能买到国际象棋博弈程序，没有人称之为人工智能。它只是个会下棋的小算法。

首先，有个定义的问题。然后，它变得更加复杂。人们认为程序应该理解自然语言，但我们的话语方式对电脑或别人来说太不精确，无法弄清楚要做什么。这正是我们创造编程语言的原因。只要试过以英文写成的规格为基础开展工作，他们就会知道自己无法借此编写程序，因为它不够精确。因此，如果人类都不能做到这一点，基本上就不可能指望还能让机器做到。当你面对所谓的人工智能程序时，电脑必须已经掌握一个词汇表。比方说，你有5个命令，你想让机器理解与之等价的所有可能的英文表述，但是它理解不了所有等价的英文表述。有人可能会说："拿个工号。"而英国绅士可能会说："劳烦你给我们的员工派个数字编号。"这正是搞AI的人试图解决的大麻烦。

人工智能相关的许多承诺都被误解了。人工智能已经教导我们的有关语言的东西是美好的。那么，我是否认为人工智能值得做？绝对是。我是否认为这会转化出很棒的产品？会有一些。我是否认为这会实现你在大众传媒上了解到的承诺？根本不会。我会投很多钱到人工智能领域吗？没门。

采访者：前面你把金钱与权力和声望同等相待。体验过成功之后，你身上发

生了哪些自己不情愿的变化？

拉斯金：我给你说件趣事。我有过一辆1970年买的橙色旧卡车。International Harvester公司出产的。它已经在国内来回跑了好几趟，什么都装过，后车厢里还可以睡觉。这辆卡车真是太棒了。在苹果公司当经理时，我就开这辆橙色旧卡车载着同事出去吃午饭。不过我很忙，没时间跟踪汽车的最新动向。我已经好几个月没进汽车商店看车了。最后，有人开始旁敲侧击地提醒我："你怎么不弄辆好车啊？"哦，好吧，我真是个负责任的员工。

周围其他人开的都是保时捷928或奔驰，而我从未想过要开什么豪车。但是他们非常想让我开辆体面的车。于是我找我兄弟商量，他给我支了一招。他对我说："一辆全新奔驰或保时捷的价钱，完全可以买辆车况极佳的二手劳斯莱斯。这样就不会有人再抱怨你开的车不够体面，而你也不用完全照别人的想法行事。"于是我买了辆二手劳斯莱斯。说来也真逗，公司里有些人之前从来不跟我说话，但是在我换车之后，他们突然开始笑脸相迎，乐意跟我搭话。



不管怎么说，换车之后，我还真感受到了不少我们所享有的民主。当我驶入加油站时，他们会一拥而上，开始悉心擦拭车窗，并尊称我"先生"。我可以把车停在餐厅门口的禁泊区，他们会说"先生"，"谢谢您"，"我们很高兴您把车停在大门口，这样所有人都能看到有位劳斯莱斯车主大驾光临"。另外，驶入麦当劳免下车购买通道时也很有意思。

劳斯莱斯让我看尽世间人情百态。我去机场时会有5个家伙迎上来，打开车门献殷勤："先生您好，欢迎来到旧金山国际机场。"开别的车到机场时，我可从没享受过这种待遇。他们盼着会捞上大笔小费。在圣何塞闹市区的低底盘汽车之间驾驶也是妙趣横生。那些驾驶者对我这笨重的铁疙瘩投来的敬意，不亚于我对他们低底盘汽车的爱慕。

我可以直接进入Kaiser之类的大型工厂，那里到处都是警卫，径直开进去就行。我一直想到工厂里头参观一下，但从未得偿所愿。在劳斯莱斯里，我只需开过去说一句："我跟Mumford先生有约。""非常感谢您的光临。直接往里开。"

离开苹果公司后，我把劳斯莱斯卖了。

续写传奇人生

杰夫·拉斯金于1978年加入苹果公司，是公司第31名员工。起初是出版部门的主任，后来由于对于个人电脑有独创的认识，于1979年负责启动Macintosh项目，招募人马，做了许多创新性设计。该项目后来由史蒂夫·乔布斯于1981年接管。

拉斯金于1982年离开苹果公司，创办了信息设备公司。信息设备公司推出了掌上电脑Swyft，并将该项设计卖给了佳能。后者在1987年推出了界面一新的Canon Cat（佳能猫）电脑，获得广泛关注，但因种种原因，Canon Cat并未取得商业上的成功。

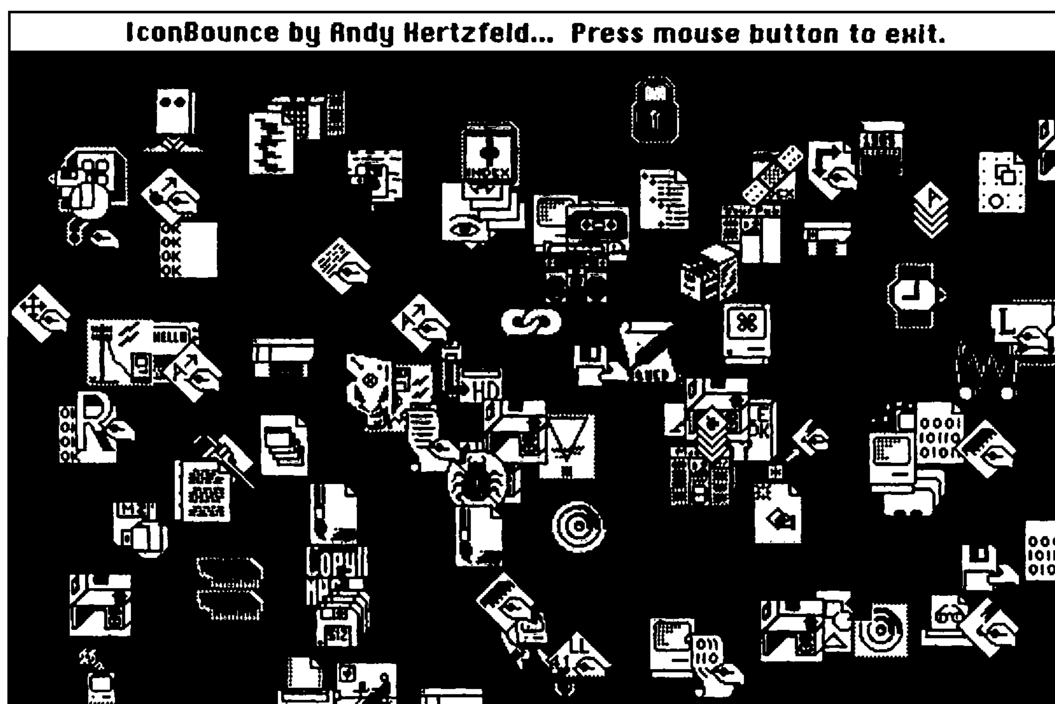
拉斯金作为湾区人机界面组的成员，经常为各种消费产品的人机界面做评审。他在2000年出版了人机界面专著《人本界面》，总结了自己的设计思想。

2000年的时候，拉斯金开始了一个新的计算机界面的设计，这种界面被称为“人本环境”（The Humane Environment）。2005年1月，拉斯金将其重命名为Archy。通过使用开源元素演绎ZUI（Zooming User Interface，缩放用户界面，区别于人们熟悉的图形用户界面GUI），系统地实现了他的人本界面概念。同一时期，拉斯金还接受了芝加哥大学的邀请，担任计算机科学系副教授，同Leo Irakliotis一起，开始为人本界面和计算机企业设计全新的课程体系。

拉斯金业余兴趣广泛。他出任乐队指挥，精通多种乐器。他制作的艺术作品曾在纽约现代艺术博物馆展出。他设计并销售过遥控滑翔机，还拥有一项飞机机翼设计专利。他爱好射箭、自行车运动，偶尔还开开赛车。

2005年2月26日，杰夫·拉斯金因胰腺癌病故，享年61岁。他留给儿子的最后一件礼物是一把设计简洁而独特的剃须刀。





运行安迪·赫兹菲尔德编写的程序，这些图标就会在你的Macintosh屏幕上跳动。只需键入附录里的程序代码即可实现。



15 安迪·赫兹菲尔德



安迪·赫兹菲尔德（Andy Hertzfeld）出生于1953年4月6日，在费城西郊长大。他上高中时开始对计算机着迷，当时写的程序有一个就是学校舞会的约会程序。高中毕业后，他进入布朗大学，学习物理、数学和计算机科学，并于1975年取得计算机科学学位。1979年，他获得加州大学伯克利分校计算机科学硕士学位。研究生毕业后，加入苹果电脑公司，从事Silentype打印机[①]、Apple III操作系统及其他产品的开发。1981年2月，他作为第二名程序员加入Macintosh开发团队，参与项目开发，并成为Macintosh操作系统的主要开发人员。最近，赫兹菲尔德离开苹果公司，开始自己单干。他后来在Macintosh上开发了Switcher[②]，另外还有一款低成本、高分辨率的数字化仪，称为ThunderScan[③]。

我步行至赫兹菲尔德靠近帕洛阿尔托市中心小巷里的小房，门开着，门口有些尘土。走进一间昏暗的房间，我听到安迪正在通电话。他坐在沙发上，光着脚，穿着牛仔裤和T恤，一条腿盘在身下，正在打电话，声音欢快而有生气。他是个身材不高但很结实的年轻人，头发有点蓬乱，举止友善而放松。

① 苹果公司推出的第一款打印机，1980年3月发布，打印机固件由赫兹菲尔德编写。

② 当时Macintosh系统软件还不支持同时运行多个程序，Switcher允许Macintosh用户同时打开多个应用程序，并在这些程序之间切换，以提高工作效率。

③ 一个低成本解决方案，通过将打印机色带替换成光学传感器，把ImageWriter打印机改装成高分辨率的扫描仪，外加独创的辅助软件，完成图像扫描。详见http://www.folklore.org/StoryView.py?project=Macintosh&story=Thunderscan.txt。

他微笑着挥手示意我进屋，指了指一把椅子，一边继续跟电话另一头通话。

我找了把椅子坐下，环顾四周。我们坐在一间开放式客厅里，客厅很大，放了一套大沙发和一把椅子。我试着打开自己身旁那盏灯，但没亮。夕阳西下，房间里渐渐暗下来。客厅里还有两个书架，堆满了书，有摞成一堆的，也有立着放的，一个合成器，一台Macintosh，还有一组靠墙放的立体声音响。房间略显凌乱，不过一点都不脏。安迪后来解释说，他其实非常懒散，不过我很幸运，因为我去拜访的当天小时工刚好来过，房间收拾得很干净。过了一会儿，安迪挂断了电话。他健谈而且友善，坦率地跟我谈论起自己的程序员职业经历，回顾自己在苹果公司的好时光和不顺的日子，并且阐述了自己的编程之道。安迪特别强调，他之所以编程，首要原因是因为编程充满乐趣，等到编程不再有乐趣的时候，他将转而追求其他东西。

*　　　*　　　*



采访者：可以具体说说你是什么时候开始迷上个人电脑的吗？

赫兹菲尔德：对我来说，1977年3月的首届西海岸计算机交易会（West Coast Computer Faire）是件大事。交易会令人心潮澎湃，因为这是计算机行业众多公司第一次真正的聚会，计算机作为一个行业那时在西海岸刚刚兴起。我第一次看到Apple II，我心说："这就是我想要的。"它真是酷毙了。当时计算机支持图形已经很了不起了，但是Apple II支持彩色图形，真是令人激动不已。遗憾的是，那些Apple II有点小贵，我还买不起。最后，到了1978年1月，苹果公司搞了次促销，每台电脑有400美元的折扣，于是我买了一台。一般来说，现实和预期总有差距，但Apple II比我预想的还要好一倍。购买那台机器是我平生最好的一单买卖。

我原本打算将自己的研究生课题定为个人电脑，因为在我看来个人电脑是世界上最激动人心的机器。令我惊讶的是，系里几乎每个教授都认为个人电脑是计算机科学有史以来最糟糕的发明。结果，进入研究生二年级之后，我对学校课程兴趣全无。我只想摆弄自己的苹果电脑。

采访者：那些教授为什么反对个人电脑？

赫兹菲尔德：因为跟他们编程用的大型计算机相比，个人电脑远没有那么强大，存储器也小得多。他们认为个人电脑使计算处理技术倒退了20年。他们完全体会不到普通人用上这些机器的那股兴奋劲儿。突然之间，电脑变得所

有人都触手可及。我猜很多学者对此毫不关心。我不在乎个人电脑的内存只及大型计算机的百分之一。个人电脑要迷人得多，而且只要一两千美元就能买到。于是我开始自学Apple II。1978年4月，我成为旧金山湾区第一家苹果电脑用户俱乐部的创始会员。在俱乐部里，你可以展示自己写的程序，也能得到其他人的程序。

采访者：当时你都写了些什么样的程序？

赫兹菲尔德：我在俱乐部里展示的第一个程序是个多功能音乐编辑器。它会在Apple II上播放滚石乐队的歌曲《妈妈的小帮手》(*Mother's Little Helper*)。我编写的第二个程序是易经算卦程序，利用Apple II的图形与用户交互。在程序中，为了模拟投掷蓍草杆，我在屏幕上画了一条线，每秒钟闪烁几百次。当我按下Apple II的手动遥控杆时，屏幕就不再闪烁。我就照这样做六次，算得一卦，然后显示对这个卦的解释。我完全被迷住了，易经和计算机的数字化本质是如此吻合。这些早期程序都是用BASIC写的，毕竟那时还是BASIC的天下。后来我开始使用汇编语言，因为我确信这是写出好程序的必由之路。



采访者：你最早是什么时候开始编程的？

赫兹菲尔德：上高中的时候。我很幸运，因为上的那家学校有台笨重老旧的西电（Western Electric）电传打字机，而且连在一台通用电气分时计算机上。1969年，高中三年级，我开始学习用BASIC和FORTRAN编程，然后是PL/1。那时我十六岁。

采访者：你就是从那个时候开始迷上编程的？

赫兹菲尔德：是的，差不多。跟很多人一样，我甚至从没听说过电脑，直到有个朋友告诉我有电脑这东西。他给我看了一些代码清单，但即便如此，对于电脑是怎么回事，我还是没有清晰的概念，直到第二年我上了课之后才算弄清楚。然后它就敲开了我的心扉，我爱上了电脑。现在电脑已无处不在。我们每天都会见到这样那样的电脑。

我发现自己在编程上有点天赋。电脑能让人体会到控制和力量的奇妙感觉。随便想点什么，然后让电脑做你想到的，这种感觉太妙了。并且一直都是如此。这正是吸引我进入这个领域的原因所在。学习编程就像学骑自行车，光看手册指南行不通，你必须亲身实践。

我最早写的一个程序是班级约会程序，并在春季舞会派上了用场。我在

编程上全靠自学，因此这个舞会程序用起来不怎么样。我叫了一帮学生，让他们聚到一起观察程序运行情况。但是我漏掉了一种情况，即同一个女生会被分配给所有男生当舞伴。有个女生跟每个男生都匹配成功，而后我意识到自己忘了一个步骤：一旦有了舞伴，就应该把这个人的名字除去。雪上加霜的是，偏偏这样一个奇特的家伙让我碰上了。

采访者：你什么时候开始认真对待编程的？

赫兹菲尔德：进入大学后，我先是主修物理，然后是数学，直到大三，我才开始思考："哦，天哪，我今后该选择什么行当？"我不想穿正装打领带，成为朝九晚五的上班族。后来，有一阵子我又担心："天哪，我可能永远也找不到工作，会一事无成。"然后，当我发现研究生数学课程真的很难时，心里又开始嘀咕："天哪，我讨厌自己的专业。"此时我原先当一名数学教授的想法开始淡出视线。我并未考虑选择编程作为自己的职业。当时编程还谈不上是种职业。



通过暑期编程兼职和学生计算设备中心3美元一小时的咨询工作，我对计算机有了更加深入的接触和了解。随后，在布朗大学上大四期间，我又修了几门更高级的计算机课程。我决定攻读计算机科学专业的研究生，但仍未考虑把计算机编程当作自己的职业追求。

当时的情况大不一样。那还是1975年，人们对计算机和编程的认识跟现在不同。个人电脑还没有让大家注意到计算技术。

采访者：当时编程意味着什么，现在又变成了什么？

赫兹菲尔德：1975年，程序员一般在银行或者更酷的NASA（美国航空航天局）工作，但作为职业还未真正成形。十几岁的孩子一般都不会想到要当一名程序员。除了在科幻小说里读到过，普通人还不知道什么是计算机。1975年，没人买得起电脑。实际上，那时个人电脑革命才刚刚开始。那年秋天Altair套件投放市场，我开始在杂志上看到介绍微型计算机的文章。这些进展对我来说非常重要。

你知道，作为程序员，我总是喜欢探究事物的底层细节，深入挖掘最基础的层面，理解计算机在那个层面的所有工作机理。我的兴趣在于操作系统，但是我被自己编程过的所有计算机搞得灰心丧气。我从来没有机会深入这些计算机的最底层，因为总是许多人共用一台计算机，而我又不在他们允许进行底层编程的程序员之列。我没法做实验。我想要是自己有台电脑，那

么想用它做什么都可以。但是Altair套件并不适合我，因为我不喜欢焊接和组装器件。不过，我立刻意识到个人电脑正是我想要的。

采访者：计算机和编程的哪些方面是你真正喜欢的？

赫兹菲尔德：我非常喜欢计算机图形。我不大喜欢小型机和大型机，因为它们与用户交互不多。最重要的是，我喜欢打动别人。我可以向朋友展示某样东西，看到他们张大嘴巴惊呼"哦，哇哦"的样子。图形和声音这两样东西大部分人都会感兴趣。其实我最想打动的是自己。

采访者：说说你的求职经历。是什么公司有意雇你当计算机程序员？

赫兹菲尔德：有个家伙飞到德克萨斯对我进行面试，他觉得我不错，就给了我一份工作，是德克萨斯州加尔维斯敦最大的保险公司。加尔维斯敦非常奇特，想不到美国居然还有这种城市。这个城市过去是旅游胜地，到处都是酒店，还有强壮的纹过身的虾船船工、石油工人、养牛人，各色人等云集于此，形成奇特的反差。我跟这些文化格格不入。



在我进入这家大型保险公司之前，公司里有个非常糟糕的家伙负责管理公司所有计算机服务。他有点自大狂。一般来说，当你负责管理电脑设备时，你总是希望自己管理的计算设备尽可能地强大。这会让你的简历增色不少，帮你拿到更高的薪水。一般来说，你的薪水跟你管理的计算机数量成正比。结果，这家伙劝说公司购买了4倍于他们实际可能用到的运算能力。他们有两台非常昂贵、价值1500万美元的IBM计算机。当他们发现自己只用得到百分之十五的运算能力时，就直接把这家伙开掉了，但这些计算机没办法脱手。巧的是，NASA距离加尔维斯顿只有20英里，他们对运算能力的需求是多多益善，于是我老板说服NASA购买了保险公司用不了的百分之五十计算机时间。与NASA的合作给我的工作平添了不少乐趣。我的工作是维护和增强NASA正在使用的APL系统。在离职进入伯克利研究生院之前，我在这家公司学到了很多操作系统方面的知识。

1976年9月，我进入伯克利学习。其实，我正是在伯克利找到了第一家电脑商店，那是大学街（University Avenue）上的一家个人电脑零售店Byte Shop[①]，我会经常到那里闲逛，只为看看那些电脑。大部分电脑都还有指示

① 最早的个人电脑零售商之一，1975年12月保罗·特雷尔（Paul Terrell）在加利福尼亚山景城开设了第一家零售店。

灯和开关。我乐此不疲。

采访者：你刚才提到，进入研究生二年级后，你对Apple II的兴趣大过对学校课程的兴趣。这段时间你在学校里的情况如何？

赫兹菲尔德：我在精神上已经辍学了，不过我还是在继续上课。当时苹果公司已经售出了5000台Apple II，我隐隐觉得这里头有商机。不过，在遇到Apple II传奇程序员鲍勃·毕肖普（Bob Bishop）[①]之前，我还没想到售卖Apple II程序。他早我6个月买的苹果电脑，当我还在琢磨这台电脑的时候，他已经用汇编语言写了几个很不错的程序和游戏。他是1978年第一个推出Apple II程序的程序员。在一次Apple II用户组聚会上，我见到了他。他跟我说他一个月挣6000美元，这让我惊讶不已；我当助教一年才挣7000美元。

我开始编写更多程序。我偏爱系统程序，着实费了番劲儿写了一个字幕机程序，利用Apple II的图形为其提供当时Apple II还不支持的小写字母。这是个复杂小巧的系统程序。另外我也开始在*Doctor Dobb's Journal*和*Micro*这类杂志上发表文章和自己写的程序。我正打算把字幕机程序发到杂志上时，刚好遇到一个朋友，他也是Apple II用户，他的名字叫巴尼（Barney），他对我说："你疯了吗？你可以挣到5万甚至10万美元。"我仔细想了想，随后我们开始开展业务。我们打算利润五五分成。我负责打造产品，其余一概不管。他负责制作、编写手册以及运营业务。当时世界各地已卖出了六七千台Apple II。



巴尼开始给其他公司展示这个程序。1978年12月，我们终于带着程序到了苹果公司。我在俱乐部里见过几个苹果公司的员工，包括我的偶像史蒂夫·沃兹尼亚克（苹果公司创始人之一，绰号沃兹），我一直渴望成为他那样的人。我们必须跟史蒂夫·乔布斯谈判。我们对这次会面担心得要死，因为我们见过两个苹果公司的早期雇员，他们说："乔布斯是个怪物，他会把你们撕成碎片。你是提着自己的命向他展示这件产品。"但实际上他对我们超好，也许好得不能再好了。总之我们让苹果公司销售我们的程序。我们跟他达成一项协议，他们每卖一个程序付我们5美元。我们约好下周拜访苹果公司敲定细节，并向工程人员解释程序是怎么工作的。后来乔布斯说："我改主意了。你们的程序与我们的产品线不符。"我并不是太难过，因为我知

① 1978年，毕肖普因在Apple I/II软件开发上的成就而受沃兹的邀请加入苹果公司，1981年离职。

道我们还可以卖给别人。最后是Mountain Hardware（现更名为Mountain Computer）把这个程序推到了市场上。

没过几个月我就挣了4万美元。我想既然自己能写出这么挣钱的程序，那就没必要再上学了。然后，到了次年1月，乔布斯打电话给我，给了我一份程序员的工作。他有个特别的项目希望我能参与。我犹豫了几个月，当我在1979年第5届西海岸计算机交易会上再次碰到他时，他对我说："噢，你怎么还不来上班？"于是我就去参加面试。面试结束后，我差点决定不到那里工作。

采访者：你为什么对到苹果公司工作不感兴趣？看起来好像就要梦想成真了。

赫兹菲尔德：因为苹果公司管理Apple II开发的负责人都是从惠普挖来的，他们对Apple II一无所知。苹果公司经历过几个不同的时期，每次都会大批雇用某家公司的工程师。它经历过惠普时期、施乐时期、DEC时期，但早期惠普占据了主导地位。公司里还有一些像我这样的Apple II拥趸，比如鲍勃·毕肖普、查理·克尔纳（Charlie Kellner）和里克·奥里乔（Rick Auricchio），他们都买了Apple II，并且对它十分着迷，自然而然就加入了苹果公司。但是，他们也聘用了那些开始上班却还不知Apple II为何物的家伙。在我看来，这真是不可理喻。既然都不喜欢Apple II，为什么还要到苹果公司工作呢？这真是个糟糕的经营决策。



一开始面试我的人有一半不愿谈论Apple II。他们只想聊UNIX，这我也能谈，我在伯克利做的就是UNIX。苹果公司拥有世上最棒的产品，可这些家伙居然都不喜欢它。不过，最后一个面试我的家伙跟我很对路，他的名字叫迪克·哈斯顿（Dick Huston），Apple II的引导ROM就是他写的。我们讨论了Apple II的磁盘驱动器相关工作细节，这些是我不懂但又迫切想知道的。短短10分钟我就学到了苦苦思索数月之久的东西。Apple II磁盘非比寻常。这是件了不起的、精妙绝伦的工程设计。大部分计算机磁盘控制器卡通常多达30颗芯片，包括若干昂贵的大规模集成芯片。而沃兹的磁盘控制器卡只有5颗廉价的小芯片，而且比其他昂贵的卡工作得都要好。后来我成了磁盘驱动器方面的专家，掌握其工作的微小细节，我破解防拷贝程序正好要用到这些知识。如果有人想把新的游戏程序拷贝到磁盘上，他们会带着程序来找我。我会跟苹果公司另外一两个人比试，看谁先破解防拷贝保护。

1979年夏天，我进入苹果公司工作。我打造的第一款产品是Silentype打

印机。原本想着进入苹果公司不仅可以了解Apple II的细节，还能了解计算机科学的奥秘，但是很快我就发现自己比公司大部分人更懂Apple II。我开始学到的最多的是股票期权方面的知识。公司里所有人都在谈论这事。我那份工作的待遇是年薪2.4万美元，外加股票期权1000股，后来提高到2000股。我没怎么把股票期权当回事。我觉得1000股不名一文。

与此同时，苹果公司正在设计Apple III。那段时期它被叫做“Sara”，不过大家都把它视为Apple III。我以为它四周会架上一圈机关枪，但实际上就那台机器孤零零放在那儿，旁边也并不是总有人盯着，这让我非常惊讶。除了开发Apple III的那些家伙，大多数人并不觉得它有什么激动人心的。工作之余，我会写些演示程序测试它的新功能。这真是令人兴奋不已，因为它是全新的硬件，一切有待你挖掘。

采访者：苹果公司其他人有机会做个人项目和研究吗？

赫兹菲尔德：在苹果公司，这种人大概有两三个吧。毕肖普就是其中一个。他给Apple II添加了语音识别，无需额外硬件，只需在磁带端口里插入麦克风。那个时候，沃兹还在苹果公司，我记得自己就挨着他的办公桌，这让我激动不已。我居然就坐在我的偶像旁边，每天可以跟他说话，一起吃午饭。仿佛身在天堂。



沃兹做的项目五花八门。其中一个是用尽Apple II的全部内存来计算数学常数E①，精确到小数点后十万位。这个计算非常困难，因为Apple II的内存容量并不大。他必须利用每一块内存，包括显示屏的内存，来保存这个很长的数字。他拿不到任何中间结果，因为所有内存都用来保存那一个数字。这个程序跑一遍用时14天。那14天里他就坐在边上，等着程序跑完的那一刻。通常试验到第6天，他的Apple II就会出现花屏并崩溃。他会从头开始。他做的项目全都这么稀奇古怪。

采访者：在此期间公司财务上有什么变化？这刚好是在苹果公司上市之前吧？

赫兹菲尔德：苹果公司的销售突飞猛进。从1979年秋季开始，及至1980年全年，公司销售额猛增。随后苹果公司于1980年底上市。突然之间，跟我共事的这些人全都成了百万富翁。

---

① 自然对数函数的底数。有时称作欧拉数（Euler number），以瑞士数学家欧拉命名。

我逐渐发现股票的分配极其不公平。你会以为谁对公司贡献最大，谁得到的股票最多。事实却非如此。那些成天想着怎么捞取最多股票的人得到的股票最多。有些工程师成天想着怎么谋取更多股票，甚至到了接近犯罪的程度；而其他竭尽全力编程、每天为公司工作15个小时的人反而什么也没得到。所有人对此都心知肚明，而上市则让这些问题暴露无遗。当你从派对中清醒过来，有些家伙身家已有5百万，而其他人只拿3万年薪，你当然会注意到。你心里不免五味杂陈。

有个故事是关于沃兹的。沃兹是公司创始人之一，不用说，他分到了大量股票。有些人一点股票都没有，而有的人却拥有这么多股票，对此他觉得很不公平。于是他拿出自己的股票，宣布只要是在苹果公司工作满一定年限的员工，都可以从他那里买到2500股。沃兹的用意无疑是最好的，但结果这反倒让那些坏心肠的人欢天喜地。事情是这样的，许多不在乎钱财的雇员并没有行使股票购买权，于是那些贪心的家伙就对他们说："好吧，我给你钱，我来买你的股份。"有些人不再认真工作，反而成了股票买手；他们令人生厌，个性糟糕，是不够格的工程师，毛孔里都滴着贪婪。在一个月内，有个家伙光靠这个捞了几十万美元。这实质上是在窃取沃兹的钱财。但这也是苹果公司上市时发生的故事的一部分。

到了1980年年底，在苹果公司工作不再那么有趣。通常你的头儿是从另一家公司挖过来的，对产品一无所知。最后，迈克·斯科特[1]意识到苹果公司聘请的管理人员都是些眼高手低的家伙，1981年2月，他在一天之内把这些人都给解雇了。工程部门一共有90人，他解雇了其中40人。这一天在苹果历史上被称为"黑色星期三"。人们都惊呆了。

采访者：*你是怎么参与到Macintosh项目中的？*

赫兹菲尔德：完成常规项目后，我会利用业余时间在新硬件上做各种各样的演示。我会进到硬件实验室，帮在那里干活的人解决难题。有个名叫布雷尔·史密斯（Burrell Smith）的年轻人对电脑非常痴迷，他本来是服务部门的电脑修理工，但是常常会到研发工程师待的硬件实验室，问这问那，什么不懂就学什么。1979年年底，杰夫·拉斯金，苹果公司的另一个传奇人物，正在做一个项目，要打造一款全新的电脑Macintosh。杰夫找到在服务部门

① 1977年2月到1981年3月担任苹果公司的第一任CEO，"黑色星期三"后他转任副总裁，并于1981年7月辞职。



工作的布雷尔，让他设计这台小巧的电脑。尽管布雷尔设计的机器和最终的Macintosh大不相同，但它看上去确实非常精巧。

之前从来没有人为它编程，因此在其成形过程中，我用业余时间给它写些小的演示程序。在机器跑起来的第一天，我加班到很晚，设法将一张麦克老鸭（Scrooge McDuck，唐老鸭的舅舅）图片显示到屏幕上，验证视频输出是否正常。这张小图片很漂亮。就在“黑色星期三”之前，史蒂夫·乔布斯基本上接管了这个项目，他让开发Macintosh的4个人搬到这个偏远的地方，所以我就再也见不到了。我禁不住想：“嘿，这里这些家伙做的东西都太普通了。我要去那边和他们一起参与Macintosh项目。”于是我跟迈克·斯科特说了自己的想法。那天下午，我正坐在自己的隔间里，乔布斯走过来对我说：“安迪，现在你去开发Macintosh吧。”我回答说：“不过我得先把手头Apple II的活做完。”“不，用不着，马上转到Macintosh项目上。”他说着拿起我的装备，搬到车里。

我很幸运，因为我是参与Macintosh开发的第5人。参与Macintosh开发的人全都特立独行，不喜欢在组织中工作。这是个激动人心的项目：一款造价便宜得令人难以置信的机器，功能比Apple II还要强大10倍。史蒂夫让所有聪明人聚到一起工作，没有管理人员。在某种程度上，管理结构完全被颠覆了，因为乔布斯自己也是我们中的一员。



采访者：参与Macintosh项目的其他主要开发人员都有谁？

赫兹菲尔德：编写系统软件的主要有五个人：我、史蒂夫·卡普斯（Steve Capps）、布鲁斯·霍恩（Bruce Horn）、拉里·肯尼昂（Larry Kenyon），以及具有Lisa电脑开发经验的比尔·阿特金森（Bill Atkinson）。杰夫·拉斯金带来了第一个Macintosh程序员，名叫巴德·特里布（Bud Tribble），他非常有才华，但在1981年12月退出了这个项目，回去上医学院。他现在拥有神经科学的医学博士和哲学博士学位。拉里·肯尼昂和我合作开发过Apple II外围设备，因此我知道他很棒。布鲁斯·霍恩也是重要成员之一。他几乎是在施乐PARC长大的，14岁就开始在那里工作，22岁时，他已经在开发Macintosh了。随后，1983年1月，另一个出色的程序员史蒂夫·卡普斯加入我们的小组。另外还有几个人也帮过忙，不过Macintosh几乎所有ROM都是我们五个人写的。

采访者：开发Macintosh都采用了什么样的策略？

赫兹菲尔德：在我看来，只有设计者和实现者是同一拨人，才能把工作做好。写代码的人就是设计代码的人。

采访者：你在开发Macintosh时相信它会大获成功吗？

赫兹菲尔德：当然。在我们看来，它比其他电脑要好得多。我们都认为它就是完美的电脑。但是，在开发这款电脑的1981年这一整年里，我们在苹果公司就是一团空气。开发人员不到20个，它也不是苹果公司产品计划的一部分。大家都认为乔布斯疯了，Macintosh就像是他回到车库时代的幻想。所有人都知道打造一款新电脑需要100名工程师，只有6个人可办不到。但是到了1982年初，当我们让这台电脑运转起来的时候，人们都说："嘿，也许这东西有点来头。"到了1983年，苹果公司知道Macintosh就快成形了。

还有一点不同寻常，史蒂夫打算把荣誉归给那些真正参与开发的员工。这个想法来自Lisa推出时发生的事情。公司为Lisa做了很多宣传，而产品110%的重要工作都是比尔·阿特金森完成的。可以毫不夸张地说，是他创造了Lisa。但是，在刊登出来的关于这款电脑的文章中，他没有得到任何应得的荣誉。这是因为Lisa项目有着令人难以置信的官僚作风，而Macintosh项目则自由而宽松。Lisa的组织架构非常复杂，共有六个管理层级，一百名程序员。杂志采访这些管理人员，他们都认为自己是Lisa的架构师，其实它主要是比尔创造的。

比尔心里很难受。这种事情在他上大学时就发生过。他做了一件里程碑式的工作，一部有关大脑的计算机图形电影，登上了《科学美国人》杂志的封面。他的教授却把所有功劳揽到自己身上，其实全部工作都是比尔做的。他非常失望，甚至打算辞职。为了让他回心转意，苹果公司最后授予他公司院士称号。

因此，乔布斯承诺Macintosh的荣誉都归我们。宣传工作在Macintosh推出时达到高潮。我们当然认为这是世界上最好的计算机，其他人的确也非常喜欢它。我的照片登上了《新闻周刊》和《滚石》杂志。我从16岁起就一直在看《滚石》杂志。

采访者：你是什么时候出于什么原因离开苹果公司的？

赫兹菲尔德：就在Mac发布之后，Mac团队开始组建管理层的时候，我离开了苹果公司。我跟新来的技术经理有矛盾，他最初是我面试和同意录用的，没想到他是个控制和权力狂。这家伙认为我太傲慢，自以为可以通过考核给我差评打击我。我很震惊。我掏心掏肺地为这家公司卖命，尽我所能做到最好，每天工作15个小时开发这个项目。Macintosh本不该是这样的，但是这家伙却进来把它搞成了这样。因此我不得不选择离开，但是我把自己生命中



整整两年献给了Macintosh，它是我最在意的。所以我一直待到Mac发布之后才离开苹果公司，开始自己单干。

采访者：编程最吸引你的是什么？

赫兹菲尔德：这是我能想到的唯一工作，可以让我身兼工程师和艺术家两种角色。它包含难以置信的、严密的技术元素，这是我所喜欢的，为此你必须一丝不苟，思维缜密。另一方面，它有极具创造性的一面，唯一真正的限制就是你的想象力。这两种元素的联姻使得编程独一无二。你必须既当艺术家又当科学家，我喜欢这一点。我喜欢在编程工作的核心中创造魔法把戏。看到魔法把戏——你的程序的核心——第一次正确工作时，那将是编写程序最激动人心的时刻。

采访者：你觉得自己会一直从事编程吗？

赫兹菲尔德：我想我会一直从事编程，但并不指望自己永远跟现在一样是个好程序员。



采访者：随着时间的推移，经验的不断积累，编程似乎会越来越容易。

赫兹菲尔德：做我现在从事的这种编程工作需要全神贯注。要同时理清所有不同的关系是种技能，这种能力会随着年龄的增长而慢慢衰退。年轻人长于全神贯注。随着年龄的增长，你会变得更加睿智，你的经验会更加丰富，你的生活会变得更好。但我认为老了就不能像现在这样做这么前沿的东西。现在我觉得自己比过去的我更好，但我并不指望会永远这样。我自认为未来几年还可以有出色的发挥。我并不奢望自己四十几岁的状态会跟三十几岁一样好，但我希望自己三十几岁胜过二十几岁。

采访者：你认为十年后计算机世界会是什么样子？

赫兹菲尔德：天知道？我就想写本小说，这是我一直想要做的。小时候我如饥似渴地找书看，我酷爱现代小说。我的偶像包括像托马斯·品钦[①]那样的作家。

采访者：你会怎么描述自己的工作习惯？

赫兹菲尔德：我的工作习惯一直很有弹性。针对不同的项目，我会采取不同的工作方式。今年上半年，我在Macintosh上开发了一个复杂的大型程序，

① Thomas Pynchon，美国后现代派小说家，代表作有《万有引力之虹》等。

叫做Switcher，这个项目难度很大。核心部分的开发用了6个星期，整个项目历时8个月。我真正开始用心做一件事情时，会把大部分工作放到晚上。在苹果公司工作时，我从来没在晚上十点之前离开过，只有在身体疲倦到需要休息时，才下班回家。我一般晚饭过后才开始真正编程，从晚上八点半一直到凌晨两三点。一旦真正开始专注于某件事，我会把它看得比什么都重要。

另外我非常懒散，缺乏自律。这里之所以不怎么凌乱，全靠昨天小时工过来整理了一番。我编程是因为我酷爱编程。我只擅长做我喜欢做的事情。幸运的是我喜欢编程，但是假如有一天热情不再，我将无法继续编程。

采访者：你觉得自己在编程上的奇思妙想会枯竭吗？

赫兹菲尔德：不会，原因很简单，因为电脑仍处于初期阶段，基本上每过5年一切都会改头换面一次。突然之间，你可以使用5倍于以往的内存，而很快规则又开始改变。

采访者：嗯，很显然，你为计算机行业做出了一定贡献。你有没有从中获得一种满足感？



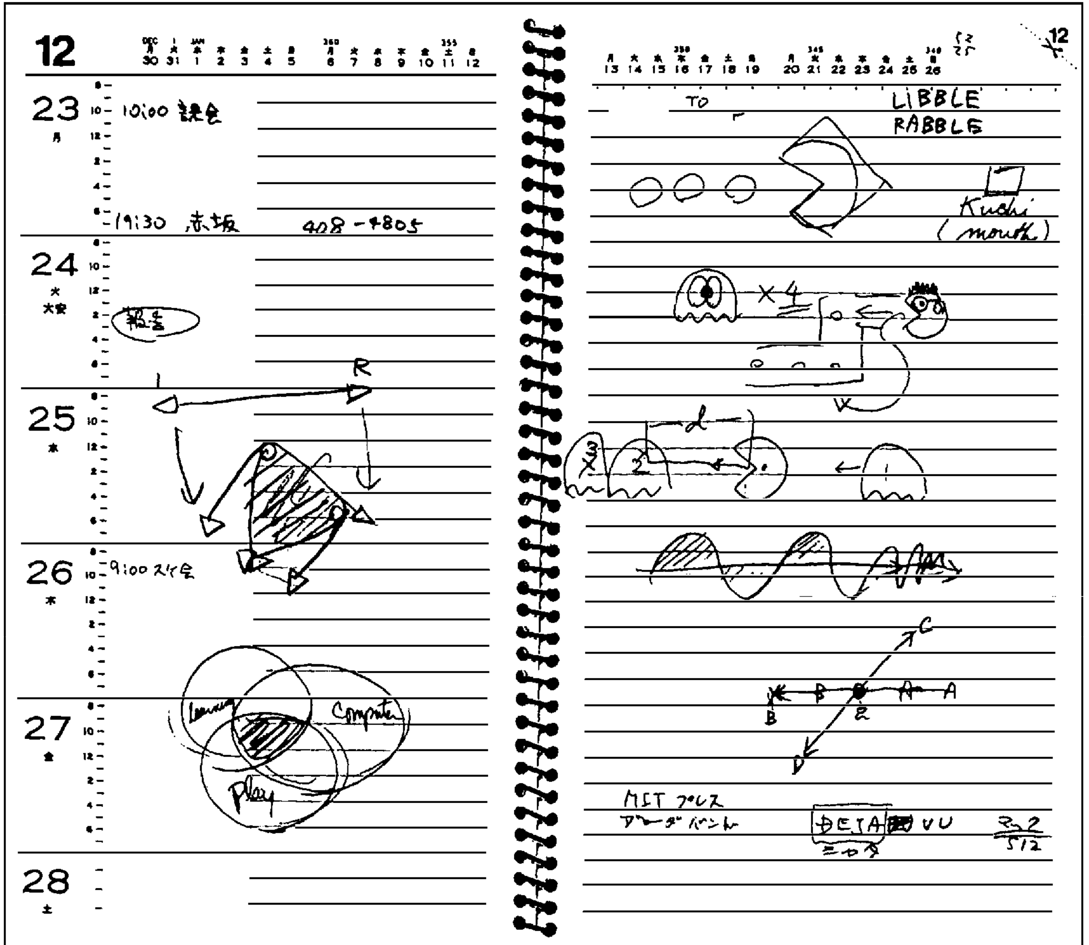
赫兹菲尔德：我看到现在有这么多人使用我的工作成果。我编写了大量Macintosh核心软件。任何时候你看到有人在Macintosh上完成了一些事情，那就意味着有人使用了我实现的一些指令，这真叫人激动不已。对我来说，激励我编程的最大动力是我想让尽可能多的人使用我的程序。

续写传奇人生

赫兹菲尔德离开苹果公司后，先后与其他人合伙成立了三家公司：Radius计算机硬件公司（1986）、开发智能手机设备的General Magic（1990）和Eazel（1999）。在 Eazel，他帮助设计了Linux GNOME桌面环境的Nautilus文件管理器。2002年的时候，他投入开源软件基金会的志愿活动，帮助米奇·凯普推广开源软件，并编写了Chandler这个信息管理软件的早期原型。

2004年，赫兹菲尔德建立了Folklore.org网站，来收集有关Macintosh开发的轶事，这些故事后来被收录进O'Reilly出版的*Revolution in the Valley*一书（中译本《苹果往事》）中。

2005年8月，赫兹菲尔德加入Google。2011年，Google推出Google+，进入社交网络领域。赫兹菲尔德是Google Circle用户界面的主要设计人。



采访过程中，岩谷在他的记事本上绘制了这些草图和图例。这些草图示意了《吃豆人》造型的演变过程，以及怪物相对《吃豆人》是怎么移动的。



16

岩谷徹



游戏设计师岩谷徹（Iwatani）1955年1月25日出生于日本东京目黑区。他在计算机、视觉艺术或平面设计方面的知识全都是靠自学的，没接受过正规训练。1977年，22岁的岩谷加入南梦宫（NAMCO），东京一家制作电子游戏的软件公司。进入公司之后，岩谷最终找到了适合自己的职位：游戏设计。他与另外四人合作，从构思到成品，历时1年零5个月的开发，完成了*Pac Man*（《吃豆人》游戏）的制作。

游戏先在日本发售，大获成功。出口欧美后，同样俘获了大批玩家。完成*Pac Man*之后，岩谷又设计了不少游戏，包括他自己最喜欢的*Libble Rabble*（《几何魔宝》）。近来，他开始更多地参与南梦宫的公司管理。

有一天，我在东京的好友和伙伴乔夫·利奇（Geoff Leach）发来一封振奋人心的电传，告诉我他通过自己的老板今泉先生获悉了Pac Man设计师的名字，他老板跟那位设计师碰巧同属一个研讨会。他觉得或许有办法帮我安排一次采访。我马上回了一封电传，表达自己热切期望做这次采访。

在日本，负责产品创作或设计的个人通常不会单列出来，并且不为外界所知。功劳一般归于团队或公司，至于个人，尽管在特定圈子里人尽皆知，但外人往往很难见到并认出来。不过，这次联络是私人性质的，因此采访得以获准。

我和利奇搭乘地铁前往东京郊外的南梦宫办公大楼。到站后，我们又穿过几条狭窄的街道，那里人流熙熙攘攘，商铺大小云集。最后我们来到一座雄伟的褐色大理石建筑前。看得出来，Pac Man大卖让公司赚得盆满钵满。

我们推开一道双扇门，走进铺着白色大理石的大堂，迎面而来的是一位机器人迎宾小姐，做着欢迎光临的手势。她全身漆成粉红色和奶油色，头戴一顶漂亮的粉红色头盔状帽子，有着蓝色的眼睛，体形娇俏。除此之外，大堂里再找不到其他人。我们停下脚步，看着柜台后的机器人。我们正要继续往大堂里走，这时她示意我们到柜台前。她旁边柜台上的计算机终端闪现“欢迎光临南梦宫”，然后提示我们查阅电话簿，打电话给想找的人。我们站在柜台前，又好笑又有点迷惑，这时公关部主任走过来迎接我们，把我们领进了一间会议室。我们就在会议室里一边喝茶，一边等岩谷先生到来。



他走进房间，我们全都起身问候，鞠躬，交换名片。岩谷先生身材高大，相貌堂堂，举止文雅而又不失阳刚之气。他身着浅黄色马球衫和粗条纹灯芯绒长裤。对方表示采访会用日语进行，由利奇先生担任翻译。岩谷说话时声音浑厚，往往斟酌再三；表述想法时，他会在记事本上写注释画草图来阐明要点。

*　　　*　　　*

采访者：你是怎么开始对计算机产生兴趣的？

岩谷：坦白讲，我对计算机并没有特别的兴趣。我感兴趣的是创造能与人沟通的图像。计算机并不是唯一使用图像的媒介，我也可以采用电影、电视或其他可视媒介。只不过我恰好选了计算机。

使用计算机也不是想做什么就能做什么，它也有限制。硬件上的局限就成了我的局限。这些因素会限制我，我跟其他艺术家一样，也不喜欢约束。我还受另一个因素的制约：最终结果只能显示在屏幕上。关掉计算机，图像便消失不见了。

采访者：你是怎么选择电子游戏作为与人沟通的方式的？

岩谷：1977年我进入南梦宫。当时我还不清楚自己到这里来要做什么。加入公司后，我碰巧选了制作电子游戏的工作。

采访者：在做这份工作之前，你学过游戏设计或者一般的设计吗？

岩谷：我没受过什么特别训练，全都是自学的。我跟一般的视觉艺术设计师或平面设计师不太一样。那时我只是对游戏设计师的角色有着强烈的认同，在我看来，游戏设计师就是要设计让人快乐的游戏。这就是游戏设计师的使命。

有一点要强调一下，我并不是程序员。我制定规格、设计特性，而程序则是跟我合作的同事写的。

采访者：*Pac Man*的设计是怎么构思出来的？

岩谷：首先，浮现在我脑海里的是日本汉字“吃”（taberu）。你知道，游戏设计往往是从文字入手的。我开始围绕这个字进行构思，在记事本上写写画画。当时市面上都是些暴力类型的电脑游戏，不是战争游戏，就是太空侵略者之类的。老少咸宜的游戏一个都没有，更别提为女性量身定制的游戏了。于是我想打造一款女性也爱玩的“滑稽”游戏。

关于吃豆人的来历，我常常会讲一个故事。有一天午餐时间，我饿得肚子叽里咕噜叫，于是要了一整份比萨饼。我只吃了一块，剩下缺了一角的比萨饼就成了吃豆人造型的灵感来源。

采访者：这个比萨饼的故事是真的吗？

岩谷：嗯，半真半假吧。日语里表示嘴巴的字（“口”，kuchi）是个方形，它不像比萨饼那样是圆的，但我决定把它弄成圆形。（参看这篇采访前面岩谷手绘的草图。）人们总是忍不住要把吃豆人的造型弄得复杂些。在我设计这个游戏时，有人建议我们给吃豆人加上眼睛。最后我们还是放弃了这个提议，因为一旦加上眼睛，我们肯定还想再添副眼镜，也许还有胡子。这样下去就会没完没了。

游戏基本构思的另一部分是食物。在最初的设计中，我让玩家置身于满屏的食物中。但是转念一想，这样一来玩家就会不知所措，游戏的目的也会显得模糊不清。于是我造了一个迷宫，把食物放在里头。这么一来，不管谁玩这个游戏，一眼就明白这是要穿越迷宫。

日语里有个俚语词“paku paku”，表示吃东西时嘴巴一张一合的动作。游戏名*Pac Man*正是取自这个词。

采访者：确定将Pac Man做成食物和吃有关的游戏之后，下一步怎么做？



岩谷：嗯，游戏光是吃东西还不够娱乐，为此我们决定引入一些敌人，给游戏注入点儿刺激和紧张。玩家必须与敌人战斗才能得到食物，而每个敌人都各有特点。游戏中的敌人是四个小鬼怪，颜色不一，分别为蓝色、黄色、粉色和红色。用这四种颜色主要是为了取悦女性玩家，我觉得她们会喜欢亮丽的颜色。

为了给游戏增加点紧张感，我想让怪物在某个时段集结起来围攻吃豆人。不过，我觉得吃豆人如果不断遭受围攻和追捕，又不免太过紧张。于是我将怪物的侵袭方式设计成波浪式。他们会先攻击，然后撤退。过一段时间，他们重新集结、攻击，再四处散开。相比持续不断的攻击，这样一波接一波的攻击来得更加自然。

接着就是吃豆人的生命力或能量设计。如果你玩过这个游戏，想必知道吃豆人也有自己的弹药。只要吃掉屏幕四个角落的大力丸，他就能展开反击，吞食敌人。除了充当猎物，吃豆人也有机会变成猎人。



采访者：*在你心目中，吃豆人是个什么样的角色？*

岩谷：即便是对日本人，吃豆人的角色也很难解释——他是个天真无邪的角色，他没有受过辨别善恶方面的教育。他的行为举止更像是小毛孩而不是成年人。你可以把他想象成一个从日常生活中学习的孩子。如果有人告诉他枪支不是好东西，他可能会立马冲出去，看到枪支就吃。但他很可能什么枪都吃，就连警察的手枪也不例外。他天真无邪，所以不分青红皂白。但是他会从经验中学习，他会了解到有些人比如警察应该有手枪，自己不能看到手枪就吃。

（岩谷开始在他的记事本上绘制带点的曲线图。参看这篇采访前面的插图。）

采访者：*设计这款游戏时最困难的是哪部分？*

岩谷：吃豆人的四个敌人小鬼怪的算法，要把他们的所有动作安排妥当。这部分比较棘手，因为怪物的动作相当复杂。这也是游戏的核心。我想让每个怪物各具特点，拥有自己独特的行进方式，这样一来，他们就不至于只是排成一列追赶吃豆人，既无聊又单调。其中，红色的怪物叫Blinky[①]，直接尾

① 吃豆人游戏中，四个怪物分别叫Blinky、Pinky、Inky、Clyde，它们分别为红色、粉色、青色、橙色，特性分别为Shadow（尾随）、Speedy（快速）、Bashful（腼腆）、Pokey（迟钝）。

随在吃豆人身后。第二个怪物处在吃豆人嘴巴前方几个点的地方，那就是他的位置。如果吃豆人身处怪物A和怪物B的正中间，两个怪物就会独自行进，形成“两面夹击”吃豆人的态势。其他怪物移动起来则更随机些。这样他们就会以比较自然的方式去逼近吃豆人。

一个人持续不断地遭受这样的攻击，就会变得士气低落。因此，我们设计了波浪式进攻——先攻击然后撤退。过一段时间后，这些怪物会重新集结，再次发起攻击。渐渐地，波浪曲线的峰谷越来越小，怪物的攻击越来越密集。

采访者：*在Pac Man设计团队中，还有其他人和你一起工作吗？*

岩谷：一个硬件工程师、一个音效制作和一个机台框体设计师[①]，加上程序员和我，真正参与游戏开发的主要就我们五个人。

这个游戏从构思到上市大概用了1年零5个月，比往常用时要长。在开发过程中，游戏的每个特性我们都会加以测试。如果不好玩或者不能增加游戏的复杂度，我们就砍掉不要。



采访者：*Pac Man受女性欢迎的强烈程度是否如你所料？*

岩谷：是的。不仅*Pac Man*大受女性玩家欢迎，而且像*Ms. Pac Man*[②]等衍生版本也非常成功。*Pac Man*在世界其他地区的流行程度出乎我的意料。当时我坚信这个游戏会在日本大卖，但是在美国和其他国家也这么畅销，确实让我大感意外。

采访者：*Pac Man有没有哪些方面是你现在希望可以改变的？*

岩谷：*Pac Man*是我很久之前完成的作品。设计这个游戏时，我自认为已经倾尽自己所能，也充分发挥了其他人的能力。当时我非常满意。不过，它跟现在的我或者我目前做的事情实在是没什么瓜葛。

*Pac Man*完工后，我又开发了*Libble Rabble*。这个游戏很有意思，主要归功于它的构思，它甚至比*Pac Man*更胜一筹。但是这个游戏的表现没有我预想的那么好。

---

① 常见的街机由框体和机版两部分组成，框体由机器的外壳和内部电子元件构成，机版是由专业的游戏厂商制造的电子基版。

② Midway出品的*Pac Man*山寨续集，后来获得南梦宫的正式授权，详见http://en.wikipedia.org/wiki/Ms._Pac-Man。

采访者：你的生活因为设计了*Pac Man*有什么改变吗？

岩谷：我的生活并没有什么大的改变，不过我对自己想要实现什么目标有了不同的认识。最近，我发觉自己很想让游戏玩家流泪，让他们体验到不同以往的情绪，跟过去玩的电子游戏不一样。我想制作一款非常戏剧性的游戏。我希望玩家有机会体验其他情绪，比如悲伤。他们不会因为受伤而流泪。玩我的游戏就像观看E.T.（《外星人》）之类的电影，他们会因为心灵被触动而流泪。他们心甘情愿地观看悲伤的电影，因为他们喜欢被感动，即使这感受令人难过。我想制作一款能以这种方式感动玩家的游戏。

采访者：你认为让人悲伤要比让人快乐更加困难？

岩谷：难得多。讲几个笑话也许很快就能把大家逗笑，但要让人们落泪却需要创造特别的情境，需要的时间也更长。要制作一部像E.T.那样让人又笑又哭的电影相当困难。



采访者：你对开发游戏有没有感到过厌倦？

岩谷：现在，我不再负责设计过程，而是更多地参与管理。这感觉不赖，因为我可以安排员工做我不太喜欢做的事，不用再经历以前有过的沮丧。另外，我还可以做自己想做的事，没人会拦着不让做，我觉得很自在。

采访者：除了设计游戏，你还做其他设计吗？

岩谷：我觉得一个人的所有行为都是在设计作品。打个比方，设想你遇见一位女性，想方设法想取悦她。你应不应该送她礼物？送什么样的礼物？最好选择什么时机送给她？你会搜肠刮肚，想出一些对策或策略。看到她脸上的幸福表情之后，你才会感受到快乐。这跟游戏设计没什么两样。

我加入了一个约莫40人组成的研讨会，我们探讨新媒体，包括教育软件以及它能解决的教育问题。这是我们必须关注的领域，因为日本的教育体制太差劲了，这着实令人难堪。坦率地说，我认为如果教育缺少乐趣、无法令人愉快，人们就不会乐于学习。

当然，我的专长是娱乐大众。如果有个学习目标能以有趣好玩的方式表述，那就可以借此开发出一款优秀的游戏。此外，从经济生存方面考虑，我对教育软件和计算机辅助教学（CAI）也颇感兴趣。只做游戏设计的公司并不能保证未来安全无虞。更何况，许多人都乐意为教育软件买单。

采访者：要想取得成功，游戏设计师必须具备什么样的技能或理念？

岩谷：你必须洞察人们的心灵，并拥有足够的创造力，想象出别人想不到或者无法想象的东西。你必须逼迫自己做些不同寻常的事，追求与众不同。你还必须有能力想象并给出组成游戏的图像，你不应该屈从于脑海里一开始浮现的简单想法。说到底，你必须乐于给人们带去快乐。这是成为优秀游戏设计师的基础，也是成就优秀游戏设计的必由之路。

采访者：你对哪款游戏评价最高？

岩谷：好吧，不是我自夸，不过我觉得*Libble Rabble*当属第一。当然，在其他公司的产品中，Atari[①]的游戏也还不错。

采访者：你认为十年后游戏设计会是什么样的？

岩谷：它会更接近电影，在现在的大型游戏开发中，该趋势已初现端倪。此外，像*Mega War*[②]之类的多人网络游戏也会不断涌现。和陌生人对战其乐无穷。跟其他人并且是你不认识也看不到的人一起玩，想想就很有意思。



续写传奇人生

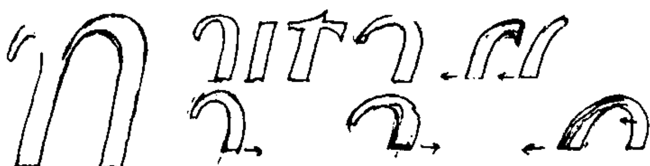
岩谷徹后续又开发了一些游戏，包括他自己最喜欢的Libble Rabble（《几何魔宝》），但都未取得像《吃豆人》那样巨大的成功。不过说到《吃豆人》，岩谷徹曾表示并未因这款游戏得到过任何奖励和表扬。岩谷徹后来在南梦宫公司内部不断获得晋升，直至负责整个公司的管理。

2005年4月，岩谷徹开始作为访问教授，在大阪艺术大学教授角色设计课题。2007年5月，岩谷徹离开南梦宫，成为东京工艺大学的全职讲师。

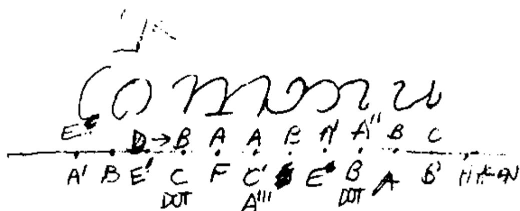
① 雅达利，美国Nolan Bushnell 1972年成立的电脑公司，街机、家用电子游戏机和家用电脑的早期拓荒者。

② 详见http://www.resistanceforces.com/。

communi-
~nication

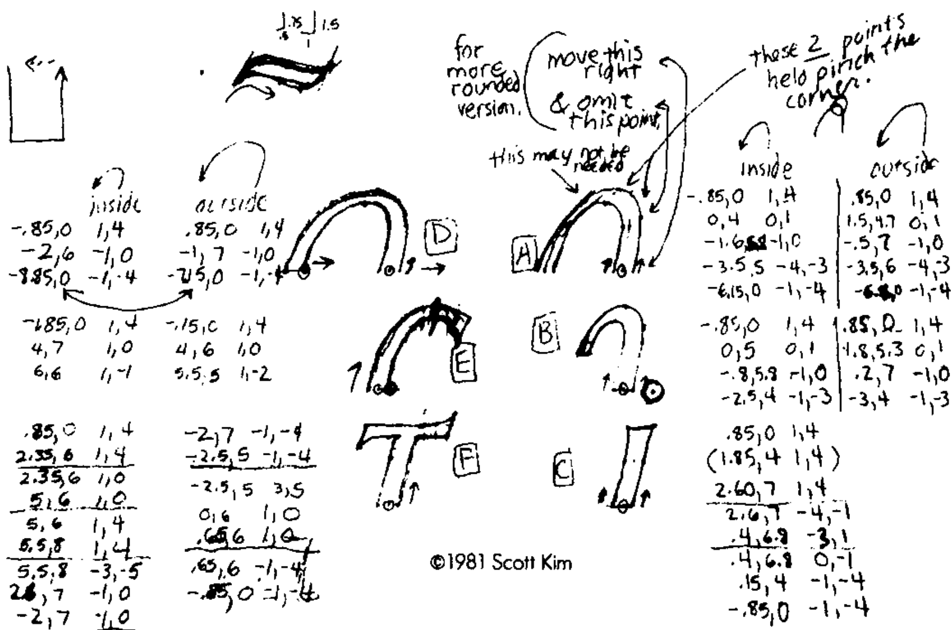












在这张草图上，很多的绘图点沿曲线和字母设计组成组合件，进而形成了字符的形状。由于形状的对称性，许多地方都是重复的。形状是先在纸上设计好了的，然后再在屏幕上做互动式调整。最终图像是由一台激光打印机生成的。附录中还有金提供的更多草图。



17

斯科特·金

斯科特·金（Scott Kim）是洛杉矶本地人，出生于1955年10月27日。他曾就读于斯坦福大学，学习数学、计算机科学和音乐，并于1977年获得音乐学士学位。1975年，在学习了图形设计的课程后，金开始从事他的“倒置”（inversion）工作——他用“倒置”这个词表述他对文字艺术的演绎，文字可以从许多不同的方向看，类似于回文和字谜。1981年，斯科特·金出版了*Inversion*（倒置）一书。很多计算机界的杰出人士对此书做出了贡献。前言是侯世达（Douglas Hofstadter）写的，他是金的老师，也是金的朋友。约翰·沃诺克帮忙对图像进行编程。高德纳和戴维·福克斯帮助排版，杰夫·拉斯金写了“后记”。金最近还开发了一个“第四方软件”，叫Inversions for the Macintosh（Macintosh使用的“倒置”软件）。该软件运行在MacPaint软件之上，包括用于创建“倒置”的许多练习、技巧和游戏。

金把他熟悉的图形设计和字体方面的知识应用到了他在计算机科学方面的工作上。目前，他正在研究一种全新的用户界面设计方法，这是他在斯坦福大学攻读博士学位的课题。早在读本科时，他就一边学习，一边在施乐公司的帕洛阿尔托研究中心作志愿顾问。他还在帕洛阿尔托从事过信息家电的工作。此外，他有自己的公司——LOOK TWICE公司。

坐在大会议桌上的Macintosh计算机旁边，我和金就他的工作和观点谈

论了几个小时。他是一个言辞温和、深思熟虑的思想者，一个致力于让计算机变成更直接、更有效的工具，能为每一个人使用的专注的学者。他有图形设计、数学和音乐方面的知识，这种不同寻常的背景可以帮助他从可视化的角度，以一个迥异于大多数工程师的方式来处理问题。金首先是一个视觉思想家。他当场就能顺手做出来的“倒置”，充分证明了他的聪明和灵活。有一天他也将成功实现他的梦想——让计算机以直接的方式工作，在屏幕上和内存中使用的是同一符号。

* * *

采访者：你是怎么想到把“倒置”作品出版成书的？

金：我在斯坦福大学读本科的时候，发生了三件大事，这三件事都启发了我出版这本书：我开始使用计算机工作了，我遇到了侯世达，我学习了图形设计课程。



1975年，我开始学习音乐系开设的一系列计算机音乐课程，那是很好的课程，但更重要的是，那些课程是在斯坦福大学的人工智能实验室里上的。这可能是一个普遍现象，人们学会使用计算机往往不是通过上课，而是通过一些朋友间的联系或彼此之间的相互帮助。就我而言，斯坦福大学的人工智能实验室是我学习计算机的发源地。那是一个非常好的学习环境，地理位置上与校园的其他部分是隔绝的。它在一个破败的建筑里，位于几英里远的山脚下，放眼望去，尽是青山、绿树、蓝天。

采访者：就是那个CCRMA（音乐和音响计算机研究中心）吗？

金：对。当时CCRMA和斯坦福大学的人工智能实验室是一样的。搞音乐的人习惯于做一个小的边缘附属物。

采访者：可以说你是通过音乐认识计算机的，对吗？

金：实际上我第一次接触计算机是在高中，但在斯坦福上大学时，我才真正开始学习有关计算机的很多知识。那一系列的计算机音乐课程让我接触到了一群非常优秀的人，他们完全出于热爱而穷尽心力，他们会花整整一个下午的时间，恨不得马上让你了解他们在做些什么。那是一种奇妙的学习方式。

令人兴奋的还有他们那里的字处理器和电子游戏。那些东西在当时还不多见。人工智能实验室当时已经有了星球大战游戏的早期版本，我们常常偷偷地跑到那里玩游戏。那些年，人们家中还没有计算机，更别提玩计算机游

戏了。那是一段非常愉快的时光，是我在斯坦福第一学年的时候。

采访者：你是否痴迷于其中？你在那里学的一切都是为了学习吗？

金：在某种程度上是的。我开始疯狂地学习计算机课程。我在第一学年结束时学了LISP语言的课程。在第二学年，我上了高德纳讲授的一系列有关数据结构的课。我学习了所有的编程语言，我大部分的计算机科学教育都是在头两年得到的。这样到了1975年，我去学习计算机音乐时，已经是一个老手了，或者至少说我是喜欢它的。

采访者：你什么时候产生了做“倒置”的想法？

金：1975年，当我还是斯坦福大学的本科生时，我上了一门艺术课程，那也是我上的唯一的一门艺术课。是一门基础设计课程，但却在几方面促进了我的发展。它激发了我对视觉方面的兴趣。我一直对视觉游戏感兴趣，尤其是动画，一直吸引着我的兴趣，不过那始终只是一个爱好。在做这门课的作业时，我创建了“倒置”。



那个学年我还遇到了侯世达。他当时正在写《哥德尔、艾舍尔、巴赫——集异璧之大成》[①]一书。我非常荣幸，在他写书的那几年里可以和他在一起工作。

采访者：你都为侯世达做了些什么？

金：我们成了朋友，并由此开始并肩工作。我们会在放计算机终端的房间里熬夜，讨论彼此的想法，然后他会打印出一个对话的最新记录，我们会根据记录讨论出更多的想法。我们就像同事。和他一起写他的书时，让我对出版我自己的书充满了信心。他的书都是他亲力亲为的。他不隐瞒任何东西。写书以外，他甚至费尽心力自己去做排版。

采访者：是的，我注意到了。而在你的书中，我知道约翰·沃诺克绘制了图。

金：沃诺克的确帮了我。他现在是Adobe公司的合伙人，而那时他是我在施乐公司的导师。

当我还是斯坦福大学的学生时，我就做过施乐PARC的志愿顾问。斯坦福大学与施乐公司的研究实验室有着密切的联系。本来我去施乐PARC也就是玩一种叫做Metafont的字体设计语言。但最后我开始用沃诺克做的JaM语

---

① 商务印书馆1996年出版。

言来制作图像。JaM是PostScript语言的前身，这个名称代表的是John和Martin，也就是这种语言的开发者沃诺克（John Warnock）和马丁·纽维尔（Martin Newell）。

当我为*Inversions*一书画图时，我探讨了是否可以使用JaM语言来做图。我并不一定非要用计算机做设计，事实上，其中有三分之二的图没有使用计算机。但对于有些设计，例如Infinity螺旋，我知道如果刻意用计算机实现的话，最起码可以看看会发生什么，这也挺有趣的。因为编程对我来说很轻松，所以没什么问题。在做Infinity螺旋时，我写了个程序来创建这些字母。我首先在图纸上画好，画了所有的草图，对各个点编了号，编写了一个简短的程序画出我设计好的单词“Infinity”。然后，我告诉沃诺克我想让他做些什么，他就编写了一个程序把直线转化成了螺旋。他有非常好的图形设计感，跟他在一起工作的感觉非常棒。如果没有计算机，这些图像的创建将会很困难，而现在通过程序把它们做出来了。从那时起，我一直在鞭策自己去尝试，看看可以对计算机和可视化表达做些什么。



采访者：*你目前正在开发的软件是用来处理“倒置”并帮助人们学习如何制作“倒置”的。你是如何着手开发这个软件的？*

金：那叫Inversions for the Macintosh，是和这本书一起发布的软件。书和软件将会打包在一起。我提出的基本问题是：“为了让书的内容更丰富，我能在Macintosh计算机上做些什么？”现在，我直觉地感到不能只是展示一些东西，然后说：“看看这个，是不是很棒。”而是应当说：“看看这个，很棒吧？你也能做到。”而这正是Inversions for the Macintosh想要实现的。我开发的这个软件是运行在MacPaint软件之上的。

编程对我来说很容易，但在开发这个软件时，我决定采取一种不同寻常的策略，不编写任何程序。在试用了MacPaint软件一段时间后，我意识到，真的用不着编程。我对编程的感受是，如果可以的话，一般都避免去编程。

采访者：*为什么你要避免编程呢？*

金：因为编写一个大型复杂程序的行为是间接的。如果可以的话，我会采用更直接的方式。我认为这是有挑战性的。在没有程序的情况下，在计算机上如何实现交互？

Inversions for the Macintosh就是我所说的“第四方软件”。这样的例子已

经非常多了。有电子表格模板，还有MacPaint软件内的剪贴画。对于人们确实需要的东西，模板是一个不错的直接回应。剪贴画是另外一个媒介的延续。虽然有用，但它显然并未采用真正的新方法去使用这种媒介。

我的软件只是MacPaint文件和字体。这和其他人用MacPaint创建的文件是完全一样的，都是靠着打字和绘图。这些文件包含视觉拼图和练习，从简单的开始，越往后就越难。但我建立在MacPaint之上的软件只是这个软件的一半，其余的则是在你脑海中所能想象到的。我非常愿意鼓励人们头脑变灵活些，在做事之前先试着想象会发生什么。想象这种技能通常在学校里是学不到的。通过在家里玩积木，孩子们会得到想象空间，但学校里是不教这些的。

采访者：你提到了第四方软件。请介绍一下，并说明你喜欢它的哪些方面？

金：第三方软件是由软件公司使用常规编程语言开发的，而第四方软件是由用户自己创建的，无需编程，至少在通常意义上可以这样理解。它建立在第三方软件之上。这就需要有非常丰富的第三方软件作为基础。



今天的软件是有问题的。大部分软件都很贵。从长远看，这么高的价格是不合理的，但现在却又有很强的势力要维持软件的高价位。软件市场还不够大，软件生产成本很高——还有其他一些观点。

第四方软件的优势是，任何有计算机的人都可以来创建它。可以在现有软件的基础上创建。有些软件是专门为此而设计的——比如游戏Pinball Construction Set就是一个例子。

当今的软件是由程序员编写的。我们将软件业与图书业作个类比。如果写书的只是造纸厂和印刷厂，我们会有很多课本和参考书，却很少会有小说和自传。这种情况的解决方案不是教大家如何印刷自己的书，而是让印刷技术变得低廉可用，这样作者就可以专注于书的内容了。编程也应该是很简单的，所有用户都可以创建自己个性化的软件。

采访者：你认为创建软件的过程将会简化到几乎任何使用计算机的人都可以创建自己的软件吗？

金：大体来说，我期望编程的本质会发生变化。现在，所有程序的构思和算法都在你的脑海中，你躲在角落里，熬夜工作，试图归整这些构思和算法，并把它们放到一个巨大的拼图中。我所期望的是，几年后，程序所需的很多

模块都是可以得到的，至少是那些常用的模块是可以得到的。举例来说，如果你想构建自己的程序，你会到一个像Radio Shack那样的商店，买预制件来构建程序。

总的来说，我所期望的是大多数人都不用像我们今天通常理解的那种方式编程。即使现在，我也觉得在计算机上编程和使用计算机是没有什么本质区别的。这两种活动差别很大，但却是一致的。向计算机输入姓名也是一种编程。另外一种编程方式要使用当今可用的编程语言，那是一个更为间接的活动。当有了可用的、更直接的编程语言，几乎任何人都可以编程时，你就不会觉得像是在编程了，也不会再称其为编程语言了。

采访者：那你会把计算机语言称为什么呢？或者说，你甚至不认为它们是计算机语言，不会意识到这是一种语言？它们会变得像英语吗？

金：我们现在用来形容这一切的词语将会改变。艾伦·凯（Alan Kay）的愿望是，当我们停止使用“计算机”这个词汇时，计算机就成功了。他常说，在电力发展的早期，一个普遍的看法是，将来有一天，在每个人的家中都会有电动马达。这就像是今天的计算机。当计算机变得足够小、足够便宜时，它们在周围的环境中就不显眼了。你甚至不会指着它说“是一台计算机”，因为到处都是计算机。除非计算机在周围环境中被忽略不见，否则就算不上成功。



采访者：你的“倒置”工作和你的博士研究有关系吗？

金：我同时在做好几个项目。“倒置”会处理通过计算机媒体展示的图形图像，只是因为这一点，才间接地涉及我的博士研究。对于我的博士论文，我严格贯彻艾伦·凯的图形界面设计精神，只是研究方向不同。

我的关注点是非常基础的。我会说：“设想我们真的从零开始，抛开我们对计算机的一切认知，并且完全按照以视觉为导向的人所喜欢的操作方式来设计一台计算机。”我所想要的计算机，就好像是一张纸。比方说，8.5英寸宽，11英寸长，这么薄，大概如此。当然不可能完全像纸，但纸张真正可爱的地方是，当你在上面写什么或画什么时，不必像你用计算机时那样去想：“如何转换到文本格式，是不是一定要按一下这里才行呢？”纸张是非常直接的，你在纸面上看到什么就是什么，而用计算机时你要想想页面背后是什么，在显示屏的后面都在做些什么。计算机就像是一张背后隐藏着电线和硬件的纸张。

采访者：你如何解决这个把计算机做成像一张纸的问题？

金：我首先思考的是现在的计算机中有哪些地方是我不喜欢的。然后我会想想图形设计和图像处理，想想我喜欢其中的什么地方，然后找出一种方法把这二者结合起来。有些视觉艺术家肯定是使用计算机工作的，他们大多使用绘画程序。绘画程序是很直接的，非常像是在纸上画。但计算机增加了一些很好的功能，画完后还可以改变颜色，一切都是可修正的。你可以随意组合图像。你可以一直不停地画下去。

很多音乐家不讨厌编程，能够编程的音乐家比能够编程的其他艺术家要多得多。音乐家，至少是受过古典音乐训练的，都习惯用抽象乐谱来工作，他们已经习惯了这一间接的过程。这不仅仅是乐谱不同于音乐，而且在他们头脑里的概念也是不同的。他们先在头脑中有了音乐，然后再用乐谱把音乐写出来。

而其他艺术家习惯于更为直接的工作方式。就这一点而言，进行即兴创作的音乐家和视觉艺术家是类似的。即兴创作更多是与工作风格相关，而与视觉、听觉关系不大。也有一些视觉艺术家是非常间接的。因为平面设计师要经过排版并向他人发出指示，所以他们不像画家那么直接。但大多数艺术家并不习惯于间接工作，他们关心的是视觉效果如何。而我们所设计的计算机，并不会让你觉得其设计者会在意事物的外观。



最基本的，我所思考的是计算机是什么，为什么我们要让计算机影响我们的生活。今天的计算机有很多问题。我并不需要马上知道解决方案，只是想再次以挑剔的方式仔细审视计算机，因为我觉得与此同时很多的成功前景会显现出来。

采访者：*那你认为计算机应当是每个人都可以使用的，还是专门给艺术家用的？*

金：为艺术家设计计算机会带来一部分灵感，但我最终想要的还是每个人都可以使用的计算机。没有一个单一的方向。我的工作方式是参与到几个稍有不同、但又以不同方式重叠的项目中，然后让所有这一切的想法爆发。

在我的论文中，我本来打算构建一个可视化编程语言，以为这也许能让计算机更直接些。我做了一个微型项目来测试这个特别的想法。我首先想到的是，这个语言有点像是一连串的字符，只是它用的是符号，像流程图那样排列。大多数可视化编程语言都有点像是电路。但一直以来我有一个很不好的感觉，觉得这事不大对劲。我直觉的反应是，还不如用Pascal语言来写。

我曾与许多人就此事进行过长谈，最后又找拉里·特斯勒（Larry Tessler）进行了交谈，他当时正在写一篇关于编程语言的文章，一年前发表在《科学美国人》杂志的软件专栏上。他是一名资深程序员，对苹果Lisa计算机颇有研究。他给了我一些很好的意见，说："作为程序员，在某个时刻你会对自己说：'哦，不，我得另外写一个程序。'"这就像是你的构想正在路上高速行驶着，却不得不绕一大圈，又回到了原点。我的感觉是，编程仍然是一个有趣的活动，但通常情况下却并不是我真正想做的事。

我花了一段时间以后才认识到我是多么不喜欢编程，我编程只是因为我想不出还有什么其他办法。但是我现在能够想到其他办法了，我坚持认为应当采取其他办法。

采访者：*当你意识到可以采取其他办法时，是一种什么感觉？*

金：在某种程度上你变得不耐烦了。你开始对大家说："拜托，我想看到有人能做出些创新来。"我告诉人们："不要接受事物的现状，它们不一定非得是这样的。"我变得更加不喜欢去编程了，除非是非编程不可。计算机是个非常诱惑人的机器，它总是诱惑你去做事。如果是在做文字处理，你就想改正每一个错字；如果是在编程，你就会想添加进去最新的功能。应该要知道适可而止。



采访者：*有谁对你做的事和做事方法产生过特别的影响吗？*

金：嗯，我想想。若是谈到软件，有三个人对我有影响。他们是杰夫·拉斯金、戴维·索恩伯格（David Thornburg，他和我一起做过Macintosh上的"倒置"）和泰德·尼尔逊（Ted Nelson）。他们这三人崇尚简洁——嗯，他们真的是崇尚简洁。当他们说到简洁时，他们是认真的。例如，他们认为5是最大的数字，你能够拥有的东西数量不能超过5。

戴维·索恩伯格是把这一观点写得最清楚的。他写了一本小书，名叫《零设计》（*Zero Mass Design*），如果你要去做什么事情，那前提是从一个非常简洁的设计开始，不论是打算写一本书，准备编写一个软件，还是即将开始一个需要规划的项目，开始的时候都要有一个非常简洁的设计。更极端的做法是，开始时的设计过于简单，让后面干不下去了。这需要大量的训练，因为你进入项目时就假定项目会失败。除非试过了，亲眼看到它的失败，否则你不会知道到底自己能达到多么简洁。

索恩伯格的例子，是来自詹姆士·亚当斯（James Adams）的《创意人

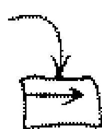
的思考》（*Conceptual Blockbusting*）一书中的“水手四号”火星航天探测器。它有个很大的、能展开的太阳能电池板。而有个问题就是，在电池板展开时，必须有一个机械装置来减缓电池板的速度，这样才不会被毁坏。于是他们尝试使用油，但搞得乱七八糟的，他们又试了弹簧，试了各种各样的东西。可是离发射的那一天越来越近了，还没找到合适的。他们该怎么办呢？请记住，问题是找到一种方法来减缓太阳能电池板的展开，或者找一个制动的机械装置。最后，有人出了一个高招，什么都不做，他们就直接打开了，太阳能电池板颤抖着，但没有断裂。如果你假设自己会遇到问题，你就会碰到问题。你必须简化、简化再简化。开始时要有一个非常简洁的设计，这会带来巨大的优势。但仍需要一个心理上的调整，你必须要有它会失败的心理准备，并坦然接受失败。

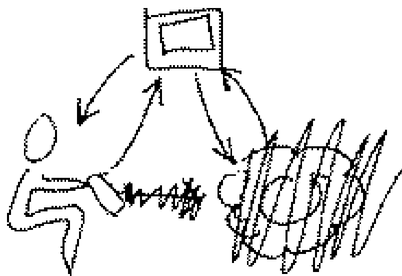
采访者：既然脑海中已经有零设计的观念了，你又如何做到把计算机做得像一张纸一样？

金：我论文中的主要观点就是在计算机科学中有这样的假定。它很微妙，所以我花了很长时间才注意到。你在屏幕上所看到的，正如艾伦·凯说的，是“用户的幻觉”。屏幕是一个非常忠实的再现，但计算机内存中是存在数据结构的，计算机真正关注的不是你，而是数据结构。它也会在屏幕上给你一个图像，但当你看这图像时，要想像一下在屏幕后面的计算机是如何思考的。苹果的MacPaint系统是很不错的，你可以清楚地看到哪个像素代表哪个像素，因此在某些情况下，幻觉是很好的，非常贴近于现实。



让我来画一个图。





这里是用户在使用键盘，这是计算机屏幕，这是内存。这种互动构成了一个小三角形。输入一些信息，进入计算机中，然后存储，并反馈到屏幕上的图像中，就是你所看到的。计算机是在内存中思考的，但你在屏幕上看到的与计算机所思考的是不同的。

人们在思考可视化编程时，实际上思考的是编程的可视化表示。编程本身并没有改变，但在它上面加上了图像。我觉得这只是表面上的包装，并不涉及实质内容。所缺的环节是在这里，在你和计算机之间。计算机看到的并不是你在屏幕上所看到的。如果计算机能够完全以用户那样的方式来处理屏幕，那么互动将是直接的。我希望看到计算机以视觉方式来思考，采取的方式就是按照用户的处理方式来处理。

我发现到这个想法实在怪异，我不得不在施乐公司做出一个例子来解释它。

采访者：你怎么可能把所有的一切都放在计算机的屏幕上？



金：哦，空间肯定很快就会用完了，所以从最真实的感觉上看，这是做不到的。目前，我只是构建了一个非常简单的程序，这种简单的程序确实可以把一切都放在屏幕上的。

在编程之前，程序必须要有生存的环境。我所构建的是一个非常非常简单的文字和图像编辑器。它和MacPaint软件属于同一类别，但要简单得多。在MacPaint软件中，在输入文字后就不能再编辑了。因为输入的字母是以位而不是以字符存储在计算机中的，所以无法再加工。

MacPaint软件显示在屏幕上的图像实际上是由位组成的。如果要编辑的话，可以拿走任何东西，即使是字母也可以拿掉一半，还可以移动它。你会忘记这些都是字母，只要当作图像就可以了，这是MacPaint软件很可爱的一点。所欠缺的是，无法从计算机屏幕背后所存储的位再还原为计算机屏幕上的字母。

采访者：你认为能够在屏幕上把一切都显示出来吗？用户会因此而觉得计算机不再那么复杂了吗？

金：我建立的那个原型不能解决这个问题，它仅仅是前进道路上的一个步骤。我是在问一个“如果……怎么样”的问题：如果所有东西都显示在屏幕上会怎么样？我认为我那个原型并不是真正的答案。但是，再说一次，人们甚至都没有把它作为一种可能性来考虑就已经放弃了。

计算机程序员通常假设在屏幕上显示的那些不是真实的。屏幕上的显示、文档等，所有这些都是附属产物。真正起作用的是算法，是在后台运行的程序。这非常具有欺骗性，在编程时，你能够处理所有那些复杂的抽象元素，你会沉浸在这种状况中。你认为它就该是这个样子。我可不信这个邪。处理图形设计让我意识到它不一定非得是这样的。当你跟一个图形设计师讨论时，不是靠说的，而是靠展现你的图片，因为那是真实的东西，那才是最重要的。你遇到平面设计师时，首先说的是："好，让我们来看看你的作品吧。"重要的不是那些充斥在每个作品中所有的抽象概念，不是那些，真正重要的都是那些画在纸上的。

采访者：*那音乐呢？音乐从未曾远离过以一个抽象的方式来演绎，是这样吗？乐谱永远都不会消失，对吗？*

金：嗯，大部分的音乐并不使用乐谱。实际上，我觉得正是因为我所受的学术训练才让我无法成为一个合格的音乐人。我被严谨的音乐理论、技巧和乐谱困住了，我不能像很多人那样即兴创作。我真羡慕他们，他们在演奏的同时还可以直接想着所有那些事情。



在50年代初期，格蕾斯·霍普（Grace Hopper），高级编程语言之母，曾四处游说人们不要用汇编语言，而是应当使用那些看起来更像英语的编程语言。人们跟她说，计算机不懂英文，所以她的想法是不可行的。但这不是重点。Pascal编程语言并不是英语，它是计算机所能够理解和人能够理解的东西之间的一个折中方案。

我希望人们能够通过更像图片的东西来使用计算机。这并不意味着我反对抽象的符号。相反，我把文字和符号当作一种特殊的图片。为了做到这一点，计算机必须能够从屏幕上的位过渡到字母，这有点像光学字符识别。

采访者：*那Inversions一书与这些又有什么关系呢？*

金：嗯，其实这中间有着很好的关联。那本书一个非常奇特的地方是，它并不严肃，而是傻傻的很好玩儿。我喜欢玩儿，很多新的想法都是从玩儿而来的。有关这本书的另一个问题是："这些是图像还是字？"这很难说。它们是两方面都沾一点。大多数人都忘记了，那些在页面上的小黑点和小白块，那些字符，实际上是由人画上去的。这些字符的字体是由人画出来的。字母也是形状，但是你忘了这一点，还认为它们是有些不同的。你忘记这一点是因为你以不同的方式制作了它们。排版和插图时必须使用非常不同的工具，

在图书的出版过程快结束的时候，它们最终合在一起了。

在古登堡发明铅活字凸版机械印刷机之前，图像和文字是同一个，是密不可分的。但后来，这两个学科独立开了。现在，我们已经有了Macintosh计算机，我看到一个能把这两者再重新结合在一起的媒介。在MacPaint软件中，文字和图像是没有区别的。

字母表中的字符实际上开始的时候是图像。它们不是空穴来风，不是从石头缝里蹦出来的，而是由人创造出来的。它们是经历了数百年的演变而变化来的。所有的符号，无论是音乐、语言还是计算机语言，都是人创造出来的，认识到这一点是很重要的。这些符号是可以改变、可以选择的。能够改变符号，这增强了人类的控制能力。

续写传奇人生



1988年，斯科特·金获得斯坦福大学计算机和图形设计博士学位。

从1990年开始，斯科特·金成为自由设计师，为网络、电脑游戏、杂志和智力玩具设计视觉谜题。他设计的谜题多达上千个，主要作品包括为Adobe.com和Juniornet.com等网站、《黑曜石》（Obsidian）和《埃舍尔之谜》（Escher Interactive）等电脑游戏、《发现》（Discover）和《游戏》（Games）等杂志以及火车历险玩具（Railroad Rush Hour）等设计的谜题。

也是在1990年，他开始为《发现》杂志撰写Boggler专栏，并在1999年成为专属专栏作家。本书访谈中曾提到斯科特·金于1981年出版的*Inversion*（倒置）一书，此后他又陆续有其他作品出版，例如*NewMedia Magazine Puzzle Workout*（1994年）和*The Playful Brain*（2011）。

斯科特·金称自己的谜题有着俄罗斯方块和艺术家埃舍尔的精神，科幻作家阿西莫夫亦称斯科特·金为“字母领域的埃舍尔”。

现在斯科特·金对网络和手持设备的谜题更感兴趣，目前正在为数学教育开发原型游戏，他还和妻子一起运营着一家名为Shufflebrain的游戏设计工作室，客户名单中有Bejewelled 2、The Sims、Ultima Online、eBay、Moshi Monsters、Family.com、Rock Band和 Netflix等。



18

加隆·兰尼尔

加隆·兰尼尔（Jaron Lanier）在新墨西哥和德克萨斯西部长大，1981年搬到加利福尼亚，一心想要寻求一种不同的生活。就像他说的那样，“在圣克鲁兹过嬉皮士式的生活，在商场里吹笛子”。结果当然并非如此。相反，在25岁时，兰尼尔就已经开了自己的公司，叫做Visual Programming Languages（可视编程语言），并且开发出了我们大多数人完全无法想象的产品。他刚进入计算机界时，是做电子游戏音频部分的程序开发。后来在Atari工作时独立开发了完整的游戏产品。他最成功的游戏是Moondust，在1983年进入了*Omni*电视节目的十佳排行榜。当前，他的程序工作围绕着一种新语言的开发，兰尼尔相信这种语言将会彻底变革计算机产业[①]。他现在住在加利福尼亚的帕洛阿尔托。

在一番搜寻之后，我发现兰尼尔的家在厄尔卡米诺路伸出的一条很小的无名土路上，而厄尔卡米诺路是帕洛阿尔托大家常走的厄尔卡米诺皇家大道上分出的一条小环路。我把车停在车道上其他几辆车的后面，然后沿着小道溜达到这座村舍风格的房子旁。这座房子漆成白色，有着蓝色的衬边。屋边草木丛生，让人感觉舒适而野趣十足，充满生气。我在前门门廊处停了一会儿，想象着会在屋里面发现些什么，然后我敲了门，期待着意外的发生。

① 公司并未成功，但兰尼尔最终推广了虚拟现实的概念。

在客厅里等待时，我看到屋里挂满了数以百计的古色古香的乐器，很多充满异国情调。咖啡桌上布满了不同大小的长笛和竖笛，大部分是竹制的。墙上挂满了包括曼陀铃在内的各种鲁特琴。我前面的墙上钉着一大块蜡染布，既不在墙中间，也并不十分平整。在房间的一角有一架直立钢琴，后盖打开着，琴弦暴露在外。靠墙边有一台雅马哈合成器，旁边则是一台Macintosh计算机。它们该是这间屋子里最现代的乐器了。我坐的地方的对面是一个书架，里面装满了各式各样的书籍，内容从佛教到Smalltalk都有。

兰尼尔穿着拖鞋跳进了房间。他穿着一件品蓝色的短袖衬衫，领口没扣，下摆在裤子外边。他是个很壮的小伙子，卷曲的头发呈淡棕色，留着胡须，浅褐色的眼睛又大又亮。他大笑着欢迎我，嗓音富有激情。我坐的是一张沙发床，而他就坐到了旁边的一张椅子里。交谈了一会儿之后，我就得到一个明确印象，他是个特立独行的家伙，满脑子的奇思妙想，让人没法捉摸。当我指着那些乐器，说起他对音乐的明显喜好时，他俏皮地说他是每周乐器俱乐部的会员。



在整个访谈过程中，兰尼尔让计算机蒙上了一层神秘色彩。他对计算机设想的很不寻常的将来，远非业界大部分人所能想象。他提出了这样的问题："如果计算机会影响你的现实世界，会影响你感知事物的方式，那怎么办？"

*　　*　　*

采访者：你现在在编程语言上做什么样的工作？

兰尼尔：嗯，基本上，我是在开发一种编程语言，比当前的容易使用得多。

采访者：更容易，是因为它使用符号和图形吗？

兰尼尔：它也需要文本，并不全部是图形。对于普通的语言，你告诉计算机做什么，它就会去做。表面上看，这听起来非常合理。但是，要对计算机写出指令（程序），你必须在头脑里模拟一个非常庞大而复杂的结构。在这个脑力模拟过程中出任何岔子，程序都会出现错误。人们在头脑中模拟庞大的结构是件困难的事。现在，我所做的就是创建可见的具体模型，用来表达计算机中运行的东西。这样，你在创建程序时就能清楚地看到它。你可以按想要的方式直接打造和改变它，不再需要在头脑中模拟程序。

采访者：设计这种特别的编程语言，你是从哪儿得到的灵感？

兰尼尔：在做电子游戏的时候，我认识到程序可以是很多不同的东西。它们可以是表达式，可以是教学工具——很多东西。我认为普通人应该可以制作程序，这种写程序的能力不应当被黑客独占。人们应当可以像谈话一样轻松地把程序说出来。在计算机里制作小小世界，应该像清晨跟朋友打招呼那样简单。我真的相信，我们会达到那种地步，那将会成为一种很有深度的沟通方式。

采访者：你的意思是，人们未来将会通过程序进行沟通？

兰尼尔：当然。设想我们是穴居人，然后有人过来，用某种方式通知我们，我们可以用一种称作语言的东西进行沟通。然后你问他："那东西有什么用？"我们今天的情况就和这个差不多。现在我们使用词这种符号，说出它们的时候就触发了脑海中的意义。但是，对我来说更有趣的事是，你可以对概念进行完整的实际建模，而不仅仅是赋予它们名字。举例来说，我们可以说"太阳系"，也可以说"行星作圆周运动"，我们可以描述这样的概念。但如果使用计算机，你可以真正创建出一个这样的系统，一个对你所描述的概念的仿真。我认为这种建模的能力，相对于仅仅使用名称来指代概念，是计算机对人类做出的最有价值的贡献。最终，这会让人们可以沟通现在还很难沟通的思想。



采访者：现在进展如何？

兰尼尔：哦，很好。我们现在的最大问题是硬件。目前计算机界的一个主要问题是：在项目开始时选择使用的机器，在项目结束时可能就已经停产了。这给我们造成了很多麻烦。我们正在开发的完全版本还要过上几年才能发布。不过，它将会改变人们对于编程的看法。所有人都会编程。我是说真的。你也会喜欢编程的，因为编程会变得很有趣。

采访者：你是针对普通大众的吗？你打算把现今使用的那些编程语言替换掉吗？

兰尼尔：现今的编程语言并没有触及到普通大众。虽然Turbo Pascal卖出了差不多50万份吧，但离触及普通大众还差得远。所以，是的，我是针对以前没

有接触过编程语言的普通大众。但是，我也同样把已经使用过编程语言的人视为目标受众。我的语言的一个特性就是它形似变色龙。技术上我不能太多作解释。它可以模拟一种传统的语言，比如，对于熟悉C语言的人，它可以样子上就像C。它会非常优雅。人们可以逐渐习惯它，而不会感到有突然的变化。

采访者：那你为什么不能在技术上对这种语言多作一点解释？

兰尼尔：因为这涉及一家公司，你知道，有商业机密的重重保护。

采访者：你的公司是Visual Programming Languages，VPL。你是怎么开始创办公司的？

兰尼尔：哦，如果你认真了解的话，会发现它真的是自然而然地发生了。我在做语言方面的工作，很多人对我很支持。有段时间，我没钱了，他们就说："让我们投资吧。"然后，喊哩喀喳，公司就诞生了。



采访者：最早是什么把你引向了编程？

兰尼尔：我有一次看到别人使用字处理器，就想："这真有意思。如果可以为音乐做一个类似的东西，那不就棒极了？"我那时候在作曲，大概五六年前吧。我觉得那是个好主意，但从来没有真的去做这样一个东西。我推迟了那个计划，来做现在这个适用性更广也更强大的工具。使用这个我们正在开发的工具，要做一个音乐字处理器或其他很多工具都会容易得多。

采访者：你在学校里学过编程吗？

兰尼尔：没有。我那时认为编程没有意思。我对计算机的概念更有兴趣。所以，我那时开始寻找那些发明计算机的人们。他们都还活着，甚至还能接受访问。你可以给他们打电话。我了解到他们过去是如何考虑计算机的。没有人一开始就坐下来仔细考虑计算机会成为什么样子。人们一开始时使用各种不同的类比方法来考虑计算机，多数是使用数学的方法。我得到的印象是开发计算机的整个过程都有点随机性。这并不是要贬低发明计算机的人们，他们确实做了一件非常非常了不起的事情。但是，将来怎样他们并不清楚。今天，我们使用计算机来做的并非他们原本设计用来完成的任务。我的意思是说，人们当初考虑编程语言该是什么时，并没有打算用计算机来进行字处理。编程语言是由具有数学思维的人发明的，而字处理是由那些具有商务和办公

室思维的人发明的，这是两个不同的世界。不同的人看待同一创意的方式并不一样。

采访者：你的意思是不是原先的创意在转变成现实的过程中经历了改变？

兰尼尔：我做的事情要追溯到五十年代。我得回去，转进每个人都略过了的一条分叉路口。今天所有编程人员谈论的都是告诉计算机做什么事情的不同方法。我的编程语言不这么做。以我的方法，你要实际看看程序在做什么，然后调整它，直至正确为止。这确实是一个不同的过程。

比如，假设你有一个做蛋糕或其他什么东西的配方。跟着做，你就可以做出蛋糕，这就像是当前大多数人的编程方法。而调整汽车的发动机就很不一样。你观察发动机运转，看它是如何工作的，然后进行调整，直至发动机按你希望的方式工作为止。我的编程语言更像后者。

采访者：你是不是说今天人们编程的方式已经没多少创造性可言了？



兰尼尔：不，不是这样。人们编程的方式很好。世上有很多伟大的程序员。我是说他们使用的语言很笨拙，他们因此而无法发挥自己的能力。特别是，许许多多的人应当去做有趣的程序，但却没有去做，因为语言限制了他们，难到让他们根本无法设想这样的程序。使用我的编程语言，人们不管在哪个领域——历史、哲学、政治、心理学，当然还有科学和数学——都可以创作自己的程序来传达他们的思想。

采访者：有哪个程序员你特别钦佩吗？

兰尼尔：哦，聪明人非常多。今天早上我还在和丹·英格沃斯（Dan Ingalls）交谈，他可是原先Smalltalk开发组的成员，很有感召力。在发明计算机的那代人中，有些人真是了不起，比如道格·恩格尔巴特（Doug Engelbart），他发明了鼠标和窗口界面，而这只是他发明的一小部分。事实上，鼠标和窗口界面可以说是施乐和Macintosh系统的基石。他现在仍旧很活跃，住在门洛帕克。还有马文·明斯基（Marvin Minsky），他差不多发明了人工智能，也很有感召力。

采访者：你对人工智能怎么看？

兰尼尔：我觉得这个术语非常怪。叫人工智能就是一件相当奇怪的事情，人

们把意识和行为联系在了一起，我说的行为是指完成某些任务中显示出来的能力。我不觉得这二者之间有任何关联。很清楚，你可以让计算机做任何你可以编程让它完成的事情。有些程序非常复杂，你会认为它们非常“智能”，但对我而言这是个没有意义的术语。麻省理工学院有人做了非常有意思的事情：教计算机识别图形。但所有的专家系统都是挂羊头卖狗肉。在很多的商业产品中，根本没有任何内容可以被称作“人工智能”。对于不同的人群，人工智能有着不同的含义。

采访者：你的语言会在商业环境里使用吗？

兰尼尔：哦，当然。它会在商业上有深远的影响。它会使软件的开发变得更快，因而也就更便宜。它会完全改变人们选择软件和设计软件的方式。当前，因为技术还很新，只要软件能工作，你就能把它卖出去。将来，软件的标准会高得多。人们会说：“当然这可以用，我可以让它跑起来。”然后他们会问：“它会不会让我感觉良好？它会不会以我希望的方式帮助我思考？它会不会在某种程度上符合我写作的方式？”我觉得很快人们就会更多地在程序的质量和美观程度上对其进行评判，而不只是仅仅基于其是否工作。



采访者：你的可视化编程语言看起来漂亮吗？它究竟是什么样子的？

兰尼尔：编程中的审美有什么用？它实际上取决于程序。有一类非常重要的程序是编辑器，能帮你处理文本，像MacPaint这样的图形程序能帮你产生图片，也有程序能帮你制作音乐。现在，如果观察这些程序运行，你会注意到它们都会假设你用它们做些什么。用字处理器的话还不那么糟，因为，对于文本，你要做的不过是把文字以正确的顺序排列起来。但对于图片和音乐你就开始看到限制了。举个例子，MacPaint不让你旋转图片，它就是缺少了这个功能。缺少这个功能本身并不是什么大不了的事，但这个程序考虑问题的方式和你的恰恰就不一样。有些人考虑的重点可能是特定的线条或线条的质量，而另一些人可能考虑的是显示的形状和色调。人们将不再寻找那个**唯一**的字处理器或图形程序，而代之以符合他们工作方式的程序。这个程序跟我思维的方式是否吻合？在我做事情时，它是否用起来顺畅？这是一种审美。

另一种审美是程序如何向你表现事物。拿前面提过的描述太阳系的例子

来说，如果教育软件可以做到那样的话，人们就会关心不一样的问题：程序把主题解释清楚了吗？程序流畅吗？程序是否允许适当的交互，让你能比看同一题材的电影获得更多的理解？将来会有更多类型的审美，现在还很难预料。当电影开始的时候，人们并不知道结局会是怎么样。计算机只是刚开始，我们完全不清楚它们将来会什么样。

采访者：你开发新的编程语言时，会事先计划好一切，还是会边开发边解决问题？

兰尼尔：在真正开始之前，我知道一点点。我会先在较小的机器上实现部分的语言，然后朝着完整的版本一步步进发。

采访者：你收集了很多乐器。你最喜欢哪件？

兰尼尔：我不知道。每周都在变。我属于每周乐器俱乐部，他们每周都会从世界的不同地方给我寄来这些有趣的乐器。实际上，乐器和计算机很有关系。它们是世界上最好的用户界面。学习乐器会给人很多启迪。



采访者：怎么在音乐中使用计算机？

兰尼尔：举个例子，我有一个程序是卡农（Canon）的编辑器。卡农有点像轮唱，人们唱或演奏同样的调子，但是不同声部在不同的地方开始，然后混合到一起。使用这个程序，你可以在一个地方输入一个音符，程序会自动帮你输入到其他声部去。你可以立刻听到整个卡农的效果。一般来说，卡农很难写，不过有了这个程序后，卡农就好写多了。

采访者：你是不是认为越来越多的人会用计算机辅助作曲？

兰尼尔：可能吧。我希望是这样。用计算机帮忙，作曲是相当容易的，只是个这么做动力足不足的问题。对于严肃的作曲家来说，计算机真的很了不起。现在，作曲家必须为不同的乐手手工复制不同的音部，所以计算机帮了他们的大忙。现在的流行音乐，你基本上只是听，但我认为越来越多的音乐会允许交互。你会真的跟音乐，跟其他人，以及跟舞蹈发生交互。我想我们很快就会看到很多这样的东西。

采访者：你有没有把你的作曲背景用到开发你的编程语言中去？

兰尼尔：很多懂计算机和数学的人也懂音乐。音乐和编程语言相似，也有一套相当复杂的记号，即乐谱。还不止这些，乐器本身更像我所要做的事情。因为，使用我的编程语言，你在程序运行时与其进行交互，而不是事先规定它该做什么，然后期望运行结果是正确的。这更像演奏乐器而不像读乐谱。

采访者：*你认为编程是艺术、科学、技能、手艺，还是……？*

兰尼尔：嗯，计算机本身没有什么特性。它们是空的，就像一张白纸。因为它们的思想完全是空的，所以它们的特性完全由人来决定，其程度超过任何其他人类活动的领域。这就是我把我的语言设计成可以具有那么多种不同形式的原因——它必须满足不同的人的需要。我更多地把编程看作是艺术。上周末我在一个电视节目上和彼得·德奇（Peter Deutsch）交谈，他说编程是一种工艺。然后，还有些人认为编程是数学。这完全取决于个人的不同看法。



采访者：*你是否对从事计算机和程序工作感到过厌倦？*

兰尼尔：哦，当然，特别是它们今天的样子。计算机是一种很恼人的机器，而编程可以让你发疯。是的，就是这样。

采访者：*你工作是在通常的上班时间吗？*

兰尼尔：你知道这个问题的答案：工作时间非常没规律。特别是有公司的这些日子。我一般在半夜工作，这样才能把事情做完。

采访者：*除了你的语言，你还做其他工作吗？*

兰尼尔：啊，全部的工作都围绕着语言。语言的某些特定部分有着不同的应用领域。比如，我们做的东西有一个子集应用于医药，我们正在为本地的医院做一些项目。

开发语言在某种程度上是非常恼人的。我很想一天24小时都花在语言的工作上，而不必分一部分时间到运营公司上。但编程工作本身很棒，工作也非常有趣。并且，如果有什么进展的话，可以清楚地看到——就显示在屏幕上面。这种进步让人感到非常满足。

采访者：*你以前做游戏的时候都做过哪些类型的游戏？*

兰尼尔：我给其他人的游戏做音乐。艺电（Electronic Arts）的有些程序用了

我的音乐。我最成功的游戏是Moon dust，它进入了*Omni*电视节目1983年的十佳排行榜，并且它带来的收入支撑了我一年。这是一个非常抽象和具有实验性的游戏，因而它的成功更加令人高兴。它里面有音乐，有小飞船飞来飞去，留下发亮的尾迹。你来驾驶飞船并控制音乐，让音乐演奏出来，但不能碰到任何错误的音符。里面有一个计分系统，但没人去注意过。我想，人们只是喜欢摆弄优美的音乐和漂亮的视觉效果。

采访者：你怎么看待计算机的将来？

兰尼尔：计算机现在非常荒谬。它们几乎不能帮人们做什么事情。在不远的将来，就会有一场乏味的市场争夺战。每个人都会集中力量争夺那些还没有大量使用计算机的大公司。但是，有些软硬件上的开发将会在接下来的数年里完全颠覆整个计算机界，它们会让每个人都感到吃惊。所以，这会变得非常有趣。

计算机界的整个结构都建立在以下假设上：计算机和程序很难制作。这就是像莲花这样的程序厂商会变得这么壮大的原因——有一个优雅的程序，并且如同雪崩一样迅速流行开来。将来会涌现大量优秀程序，并且任何人都可以制作程序。

采访者：你是说将来每个人都会自己写程序？

兰尼尔：嗯，当然大部分人还是不会，但会有很多人会。这就像今天的书籍。每个人基本上都识字，写书和不写书的人之间的区别在于是否有动力、热情和商业头脑，而不是能力问题。创作程序会是同样的情况。

并且，如果有人想要推出不是基于MS-DOS标准的新计算机，他们也不会感到害怕，因为在新计算机上做出一套软件会容易得多。今天，人们造出完整的计算机只是为了运行某些软件。将来，会完全反过来。当软件很容易制作时，兼容性就不再那么重要了。这就像把一首乐曲从单簧管转到小提琴上：你需要这儿改一点那儿动一下，但这种改动没什么大不了的。

采访者：你是不是认为计算机会越来越多地成为创作工具，而不是那么面向商业？

兰尼尔：实际上，像计算机一样，商业界也是完全由人创造的。上帝没有跑下来说，要有公司，公司里要有董事会。这些都是人为构造出来的。商业是



非常墨守成规的，它改变得很慢，所以很难说将来商业界会发生些什么。这并不一定是理性的。你知道大量的商业人士仍然在大型机上使用Cobol程序，即使这已经完全不合时宜。对此你又能说什么呢？

一般而言，我认为计算机会更多地用于创作，也会更多地用于企业。目前，计算机没为大家多做什么。也就字处理比打字好一点，数据库偶尔工作良好，特别是对于大公司。问题在于软件非常难写。我们并没有一堆不断演进、越来越好的软件，相反，当某一软件达到一定的成熟程度后，演进就停滞了。只要有东西可以工作，每个人就已经非常高兴了。

采访者：你是否认为编程语言最终会取代口头语和书面语，比如英语？

兰尼尔：完全不会。英语永远不会被编程语言取代，因为围绕着英语我们已经积累了如此多的东西。我们有莎士比亚，还有各种的词组……

采访者：嗯，那你怎么看象形文字？它们已经消失了。



兰尼尔：那是因为使用它们的人都被杀死了。但是，即使他们还在的话，也很可能在用一种象形文字的派生文字。我认为计算机会提供一种新的表达方式，人们会认识到英语和计算机各自适用于表达不同的东西。在某些领域，英语已经有些捉襟见肘。当你讨论哲学、经济学、政治中的思想时，人们几乎不明白他们彼此在说些什么。使用计算机，你可以实际建立起完整的思想或概念的交互系统的模型，甚至可以建立起思维方式的交互系统。这些都能更好地用建立在计算机上的模型来表达。英语适合用来描述，而计算机适合用来建模。在将来，这两者将会混合到一起，都会成为我们互相之间沟通方式的一部分。两者一起将会改进我们的沟通方式。无论人们在什么时候沟通，他们都更可能与对方产生共鸣。

采访者：你认为Dynabook①的概念会成为现实吗？

兰尼尔：它只是个过渡阶段，会很快过去，大概十年之内就会。

采访者：过去之后是什么？

兰尼尔：嗯，我可以告诉你，但你可能不相信。让我们看一下，现在你和计

① 艾伦·凯在1968年提出的电子书的概念，他想象这是一台可以带着跑的电脑。他所描绘的Dynabook的主要使用者是小孩，帮助小孩学习。

算机的交互仅限于屏幕。如果计算机可以影响你感知事物的方式呢？不是改变真实的物理世界，而是在你周围创建三维的对象。它们并不真的存在，但人们可以看到它们，并分享体验。人们的日常生活将包含计算机产生的图像，这是我们正在开发的技术，但是很不幸，我真的不能过多描述细节了。我知道这听起来很疯狂，可它确实会发生。

采访者：计算机会产生不存在的对象？感知模型？

兰尼尔：我知道这听起来让人感到困惑。让我们就这么说吧，它们是受到良好控制的幻觉。实际上，这并不那么古怪。应该说，相当直接了当。我可以说的一件事情是，你很可能要带上特殊的眼镜，眼镜会帮助创建这些图像。人们可以分享这些图像，并通过无线电交谈。

采访者：也就是说，我们不再在计算机旁边摆上一台打印机，而是会放上某种图像产生器？

兰尼尔：是的，就是这样。上周日的报纸上有一张漫画，画着一个黑客变成各种不同形状。你看到过吗？很有意思，这跟我的意思非常相符。我认为，最终计算机会为我们产生额外的现实空间。这种现实空间不会让我们疏离物理世界，反而会帮助我们欣赏这个真实的世界。

采访者：你觉得遗传学会和计算机靠拢吗？

兰尼尔：也许吧。可能会有光学计算机，可能会有化学计算机。生物计算机显然存在，因为我们的大脑，也许部分，也许全部，就是计算机。这是一个硬件的问题。你怎么让真正的计算机技术起作用？不过，对我而言，另外一个完全不同的问题更为重要：你会创造出什么样的文化来实际使用这种技术？你知道，技术问题本质上属于工程范畴。如果我们想要制造一个Dynabook，有人会搞明白如何去做。如果你想要基于酶或什么奇怪的东西做出密集度很高的记忆体，也许也会有人能够搞出来。但是，如果牵涉到文化，你就真的需要去创造——从虚无中创造一个崭新的世界。当你有了一个Dynabook，你会怎么去做？它和世界上的其他事物如何交互？它和普通的书籍相比如何？和午餐盒相比如何？和电子游戏相比呢？

采访者：文化不是演化出来的吗？



兰尼尔：不，它们是发明出来的。它们是人造的，可能是有心栽花，也可能是无意插柳。20世纪充满了这样的例子，因为我们发明了如此多的东西。电视以前并不存在，可是现在很多美国人花在看电视上的时间仅次于睡眠。

采访者：*但是当电视机发明出来的时候，人们并不知道他们在创造一种文化。*

兰尼尔：是的，实际发明电视机那些东西的人没有做到这一点。创造文化的是好莱坞那些做节目的人，以及那些想出怎么卖电视的人。这是一大堆人干的，而不是一两个。还有摄影，也是有人想搞清楚照片到底是什么、意味着什么，从零开始搞出来的。电影也是如此。计算机上发生的一切也都是人为的。我觉得意识到这样一个过程是很好的。我很高兴看到，自打计算机诞生起，人们就在思考计算机带来的政治和伦理影响。计算机界本身从中获益良多。我认为，到现在为止，跟电视或摄影比起来，计算机界存在着更多自我意识方面的努力。



采访者：*你对信息的力量怎么看？那是不是计算机最重要的方面？许多人都说这是计算机对社会来说非常重要的原因。*

兰尼尔：嗯，计算机实际干的事情只是操作信息，而信息是个范围相当宽泛的术语，基本上涵盖了人类的所有体验。但当人们讨论信息的力量时，我想他们指的事情相当确定，那就是我们的社会组织越来越多地依赖于物理上并不存在的东西。信息，或者以下概念中的任何一种——生活、计算机内存、一家公司的真实存在、财富、权力、地位或是工作——所有这些都可以由计算机中存储的信息来定义。我们处于一个过渡时期。之前我们生活中所需的东西来自于对物质的操作；而现在，我们开始根据信息来安排生活；最终，我们的体验将来自于信息而非其他方式。这将是一个真正的信息时代。

采访者：*物理世界的重要性不会降低吗？*

兰尼尔：不，一点也不会。就像计算机音乐并没有威胁原声乐器，摄影也没有威胁绘画一样。我相信计算机会让我们对物理世界有一个更加客观、更加充满欣赏性的视角。还有对于自然界也是这样。比如说，我认为它会触发更加强大的生态运动。仅从实际的层面上，人们将能够在不改变物理世界的前提下获得一些前所未有的经历，不会对真正的世界造成任何破坏。这是一个很大的话题。顺便说一下，我正在就这个话题写一本书，书名是*New Natures*（新自然），内容是关于如果能得到随心所欲的世界，结果会是怎样。

采访者：你认为今天对计算机感兴趣的年轻人在想着同样的问题吗？

兰尼尔：是的，我认为是这样。人们喜欢模式化，但实际上很难用一个形象来描述一群人。比如，大家一般都认为程序员穿着邋遢，通宵熬夜，等等。而当前的一代人非常倾向于Macintosh之类的东西。我认为我们还在起步阶段，人们还只是在尝试鼓捣出过得去的软件供人使用。再过一些年，生活真的会改变。现在出生的孩子将伴随着新科技成长起来，想到这些就非常令人激动。他们将真正从我们今天讨论的东西中受益。

续写传奇人生

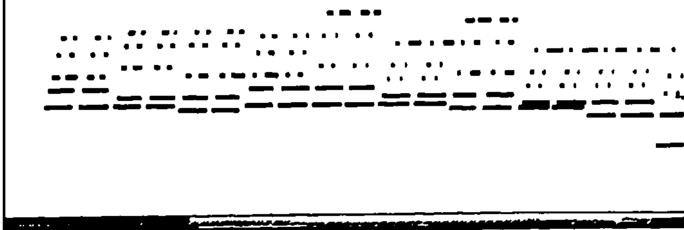
1985年，兰尼尔和托马斯G. 齐默尔曼离开Atari，成立VPL Research，致力于虚拟现实技术的商业化。VPL 创造了世界上第一个虚拟化身（Avatar）、第一个多人虚拟实境、第一个商业化的虚拟现实设备和第一个手术模拟应用。VPL一度发展得不错，但最终还是在1990年申请破产，而虚拟现实和图形相关的专利都出售给了Sun 公司。兰尼尔被认为是虚拟现实技术的先驱，“虚拟现实”这个名词也是由他所创。

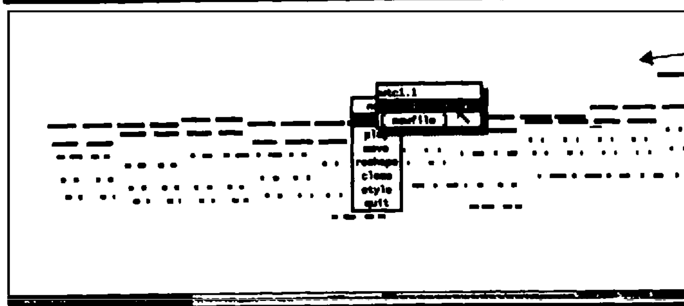


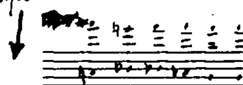
从1997年到2001年，兰尼尔担任高级网络服务公司（Advanced Network and Services）和美国国家远程全息计划的首席科学家，负责Internet2项目。经过3年开发，第一个远程全息原型于2000年诞生。从1999年到2002年，兰尼尔担任Eyematic Interfaces首席科学家。2003年到2005年，担任SGI公司客座科学家。
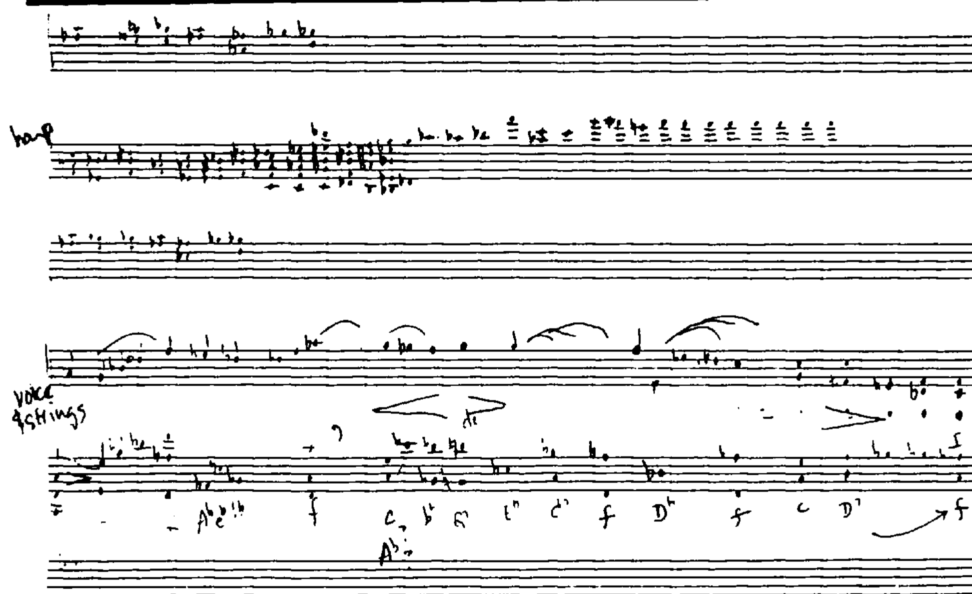
近几年，兰尼尔担任了Linden Lab公司推出的热门虚拟世界游戏《第二人生》的顾问，并先后以特约学者（Scholar-at-large）和客座架构师（Partner Architect）的身份，在微软研究院指导开发Kinect设备。

兰尼尔同样多才多艺。他是一位知名的音乐家、古典音乐作曲家、稀有乐器收藏家、视觉艺术家和作家。他从20世纪70年代后期就一直活跃在新古典音乐舞台，亦曾和他人合作为纪录片The Third Wave（《第三波》，2009年出品）配乐。他曾和多位艺术家一起表演，包括小野洋子、菲利普·格拉斯、奥尼特·科尔曼、乔治·克林顿、约翰·列侬等。他的画作曾多次在美国和欧洲的博物馆和画廊展出，他还曾帮助科幻电影《少数派报告》修正道具和情节。作为一名作家，兰尼尔曾为《发现》杂志写作专栏。他于2010年出版的*You Are Not a Gadget*一书表达了对开放文化和Web 2.0的忧虑，是一本著名的畅销书。2010年，兰尼尔被《时代》杂志提名列入百位世界上最有影响力的人。









迈克尔·霍利脑海中的新作曲方式。附录描述了霍利的音乐计算机的想法。



19

迈克尔·霍利



Droid Works（机器人工场）在卢卡斯影业公司旗下承担着把电影制作过程数字化这一意义重大的任务。作为其中的数字音频程序员，迈克尔·霍利（Michael Hawley）参与了SoundDroid（音响机器人）的软件开发，这款软件可以看作一个装在盒子里的全数字化音响工作室,它可以对声音进行储存、录制、编辑和混合，当然也可以即时回放声音。

霍利现年24岁。他成长于纽约市郊区的新普罗维登斯，并在那里熟悉了计算机和编程。在整个高中和大学的时间里，他都在附近莫雷山的贝尔实验室里打工。在贝尔实验室培育着计算机方面的兴趣时，他同时也在耶鲁大学进行着音乐和钢琴的正规学习，并在1983年获得了音乐和计算机科学的双学位。毕业后不久，他作为贝尔实验室的访问研究员去了巴黎的IRCAM（音乐与音响协调研究所）。在巴黎，他开发了计算机音乐应用程序的用户界面原型，同时又在IRCAM的音乐会系列中参加了一首为双钢琴和电子音乐所作的奏鸣曲的首场世界公演。离开IRCAM后，霍利来到美国西海岸，并加入了加利福尼亚马林县的Droid Works。

在我进入Droid Works办公室所在的大楼时，我看到灯光、摄影器材和音响器材的工作人员四处穿梭的身影。这首先就告诉我，Droid Works更多是属于电影娱乐行业而非传统的软件和计算机行业。

迈克乐·霍利带我走进了他和另一位程序员共同使用的办公室。他头发浅褐色，乱蓬蓬的，蓄着黑胡须，身着自行车赛服，戴金边蓝色太阳镜。在属于霍利的这一半房间里，他把合成器、放大器和音箱放在他的计算机终端和键盘旁边。就在这个有点拥挤和杂乱的房间里，霍利谈了他的工作和热情，谈了作曲，也谈了关于处理和编辑音乐、声轨和电影的程序的创作。在访谈中有好几次，霍利转到键盘上，播放一小段他作曲的音乐，或者展示一下他开发的软件是如何用来处理声音、创作音乐的。我得出结论，他的使命就是掌握计算机和软件的工具，从而将其应用到音乐、艺术和其他媒体上。他以一种热情洋溢、清晰而又欢快的方式讲述了自己作为程序员兼音乐家的故事。

* * *

霍利：我知道你刚从西雅图飞过来。我最近刚在西北作了一次美妙的旅行。在温哥华有一个国际计算机音乐会议。



采访者：*是的，我读到过这件事。不过我没去。*

霍利：也许你去了Digicon？不，那是一周以前。计算机会议有很多。嗯，下面就是一个故事，我保证，和编程有关。

在这儿的图形部门工作的戴维·沙利森（David Salisen）会开小飞机，所以我们认为开小飞机飞到温哥华去是个不错的主意。马文·明斯基的女儿玛格丽特（Margaret）也在这里，所以她、马克·利瑟（Mark Leather）、戴维和我钻进飞机就起飞了。对了，马克·利瑟也是图形部门的，是硬件工程师。飞行真是棒极了。我们绕着沙斯塔山①山顶转了一圈，飞过火山口湖正上方，从圣海伦火山口的蒸汽中嗡嗡穿行。坐自己的小飞机，你想去哪儿就可以去哪儿，所以我们冲到了奥林匹亚半岛，在那里做了三天的背包客。然后飞往温哥华。马克和玛格丽特无法参加会议，所以我们在西雅图放下了他们，只有我和戴维去了温哥华。

会议有点拖拉。计算机音乐的会议可能有点闷，但与会人员都很有意思，有些论文很有趣，三文鱼寿司也棒极了。就在会议之后，一个买了我的MusicDroid（音乐机器人）T恤衫的女子建议，既然我们有一架飞机，我们应该去一下温哥华岛的西岸，那里的一个偏僻的地方有温泉。那可能是整

① 和下文的火山口湖、圣海伦火山、奥林匹亚半岛都是美国西北部的著名景点。

个旅行中最漂亮的飞行了。我从来不知道温哥华岛上有这么多山。这些山如锯齿一般，层峦叠嶂，看起来有点像洛伦·卡朋特（Loren Carpenter）[①]的分形山。

我们在一个叫托菲诺（Tofino）的小镇附近把飞机降到了一条长满野草的小跑道上。从那里，我们乘出租车来到镇中心。那是一个如画般美丽的海岸边上的渔村，渔船在港口里摇啊晃啊，一座座山直插入海，一团团云雾在半山围绕。在托菲诺，我们定了一架水上飞机去温泉。他们把我们三个加一个小印第安家庭塞进一架德哈维兰海狸式飞机。我们刚飞上去之后就开始降落了，在一个前不着村后不着店的地方下了飞机。印第安人在更远的地方有块保留地，所以他们还要继续往前飞。

我们沿着破损的栈道在浓密的海岸雨林中步行了约两英里。看起来就像是《夺宝奇兵》中的开幕场景，苔藓从空中垂下，阳光在其间穿过。最后，我们到达了温泉，那儿真是非常漂亮。热水翻滚着涌出地面，流淌约两百码后进入大海。一路上，水流会经过一个10英尺高的瀑布。如果你站在下面的话，就可以很舒适地淋一个热水浴。从那里开始，水流会经过岩石间一连串的小水池，每个都能正好容下三四个人。岩石表面覆盖着软软滑滑的苔藓。每一个水池都比前一个要冷一些，所以你可以找到任何温度的水池，从跟海水一样冷直至热到你受不了。在这一区域的最下边，你可以坐在水池里，让热水冲击你的背部，就像在按摩浴缸里一样。隔一会儿，就有一个大浪涌入，让你冻得直发抖，然后，越来越多的热水会流下来，让你重新暖和起来。



我们搭起帐篷，吃了晚饭。我带了一瓶清酒，所以我们把它放在水里温了一下，然后就坐在那里，喝着清酒，仰望星空。四下无人，一切真是完美极了。第二天，我们走出去，然后飞了回来。

采访者：这跟计算机和编程有什么关系？

霍利：重点不仅仅是去参加计算机音乐会议和吸收技术，你怎么去的，可能同样有趣、同样重要。在机会来临时，你不应当浪费。编程也是如此。一路上你会发现很多的石头。在你开发程序时，新的入口、新的道路会不经意地开启。工作站和个人计算机就有点像小飞机：如果你驶往正确的方向（并且运气不错），有很多发现就在那里等着你。

---

① 计算机图形学专家，Pixar的联合创始人和首席科学家，因以分形算法在计算机上高效绘制山脉而闻名。

采访者：你在卢卡斯影业做什么？

霍利：你对卢卡斯影业及其计算机分部了解多少？要不要我来给你介绍一番？

采访者：好的。

霍利：好吧，且听我讲来。在1979年前后，乔治·卢卡斯招标寻找能把电影制作中最有趣的部分计算机化的人。留给他印象最深的是纽约理工学院的几个人——埃德·卡特穆尔（Ed Catmull）、阿尔维·史密斯（Alvy Ray Smith）等。然后他们就打包搬到了马林县，卢卡斯影业的计算机研究分部就此开张。卡特穆尔确定了三个主要的应用领域。一个是图形和图像处理，也称作图像研究；另一个是音响；第三个你可以叫做电影制作人的"字处理"。在电影业务里，人们需要一个"字处理程序"来切割图片，因为用刀片来切割胶片再粘贴到一起需要花很长的时间，就像在老式的打字机上一遍遍地打字，每次改动文字后还要重打一遍一样。



图形部门在一开始的时候就在做计算机图像的基础研究，并且他们在这方面相当自由。只有少数几次，他们需要做产品。比如，他们创作了电影《星际迷航》（*Star Trek*）第二部里的起源镜头，电影《星球大战》（*Star Wars*）系列里的一些片断，去年还做了一部漂亮的计算机制作的动画短片。最近，他们为斯皮尔伯格的一部电影《年轻的福尔摩斯》制作了奇异的镜头画面。图形部里还有一组人是做激光扫描和彩色胶片印制的。他们使用三种颜色的激光——红、绿、蓝——来对彩色胶片进行读写，分辨率非常高。这有点像常见的激光打印——你可以打开或者关闭那些小像素——不过，这儿使用的是彩色，而且可以读也可以写。激光就是Pixar使用的输入输出装置。概念是这样的，比如你想要让莱娅公主在穆尔森林里骑赛车狂奔，你拍下莱娅公主坐在蓝色幕布前的图片，拍下穆尔森林的照片，然后把它们全用激光扫描进去。一旦你有了数据，你就可以使用Pixar把图像组合到一起。因为这是一个数字化的过程，组合过程天衣无缝，不像在光学过程里那样会有那种丑陋的幕布线（matte line）。而且这个过程非常快。

图形部门造了一台非常有意思的机器，叫做Pixar图像计算机。你可能已经听说过，因为这个部门已经独立出来成立了一家新公司，名字就叫Pixar。Pixar是一台很特别的专门处理数字图像的计算机。这个机器的架构来自于图像组合的非常漂亮的算法，所以它非常高效、非常优美。我想，是卢卡斯这么说的："Pixar不仅是我们图片组合的利器，也是其他所有人想用来进入

这个圈子的工具。”对于医疗影像，对于地震数据建模，还有很多其他种类的应用，它都有着极为巨大的潜力。几乎任何可以放在屏幕上的图形数据，你都可以用这台机器处理得让人目眩神迷。图形部门一直专注在研究上，他们的总体目标一直是把最丰富的计算机影像带到电影制作中去。计算机制作的结果必须足够丰富，可以混合进有自然物像的同一场景中。

采访者：卢卡斯影业的音频部门如何呢？

霍利：我在音频部门里，我们的章程也很一般化——把任何电影制作过程中跟声音有关的一切东西计算机化。结果看来这是个很有趣的问题。如果你看一下电影制作的后期生产阶段，两到三个人就能完成图片的编辑工作了，但处理所有的声音需要一大群人。前景和背景加在一起，声音非常之多。人们可能不会意识到这一点，但如果某个声音缺少的话，人们就会觉得少了点什么。一卷卷的磁带会搬到混音师那儿去，给他们最好的原料用于混合：对话，音乐，像飞船噪音、激光爆破之类的特殊效果，甚至还有一条达斯·维达[①]沉重呼吸声的特殊音轨。每个原料都有不同的版本和表现效果。这是件非凡的工作。需要跟踪非常多的信息。



安迪·摩尔（Andy Moorer）过来领导我们部门。他是计算机音乐和数字音频方面的先驱者，有麻省理工和斯坦福的学位。他以前在巴黎的IRCAM工作。IRCAM即皮埃尔·布勒（Pierre Boulez）创办的音乐与音响协调研究所，那是一家声学和新音乐的研究机构。安迪创办了这个部门，并且开始建造一台叫ASP（音频信号处理器）的计算机。Pixar能对图形做什么，ASP就能对音频做类似的事情。一旦声音转变成了数字，你就可以随意做出各种奇妙的效果。比如，你可以把一个庞大的多轨好莱坞混音台的功能——平衡器、音量控制以及各种特效产生器——全部塞到这台机器的一个程序里。使用ASP，平衡器之类的东西不再是硬件的组件，它们只是这台机器里某段嗡嗡响的微代码。它们可以做任何你要它们做的事情。

我们在一个称作SoundDroid的系统中使用ASP。那是一个声响人士使用的字处理系统。你看着屏幕上代表不同音轨的图像，可以通过在触摸屏上点击来剪贴声音。你不需要等待倒片，也不需要等待对磁带进行剪切复制。可以很自由地把声音洒在各个地方，而不需要直接操作胶片。这是一个巨大的飞跃，对于创作过程有着深远的影响，就像字处理影响了人们的写作过程一样。

---

① 《星球大战》里的黑勋爵，著名的反派人物。

第三个部门造的机器叫做EditDroid（编辑机器人）。这个机器背后的构想是做图像剪辑的字处理。胶片的内容会转录到视频媒介上，比如光碟或磁带，然后计算机可以对其中每一帧进行随机读写。在计算机的帮助下，你只需要对实际的胶片剪切一次。使用EditDroid时，可以自由地切割并拼接胶片中的内容，实验和摆弄各种组合，然后按下一个按钮来立即预览你组装的结果。在你达到好的效果之后，再按下一个按钮，EditDroid就会输出一张清单，告诉你该在何处切割实际的胶片。

这样，Pixar、SoundDroid和EditDroid就是计算机分部的三个最明确的"产品"。

采访者：*你是怎么参与到这些东西中来的？*

霍利：在我解释之前，我应当说说卢卡斯影业的现状和将来。乔治·卢卡斯打算将注意力集中在电影制作上。他是个电影制作人，而非计算机科学家，他也不打算分散他的资金，特别是现在。因为现在他把大量的金钱投在了天行者牧场的建设上。天行者牧场将是一块制作电影的乐园。它极其壮观和美丽，像古堡一般规模宏大，隐藏在马林县北部的树林中。它即将成为电影制作者的天堂。



建造这个牧场是项费用浩大的工程。你可能已经注意到，卢卡斯影业近来没有大制作，这也是一个原因。卢卡斯影业看起来满足于把计算机部门分离出去形成新的公司。图形部将成为Pixar，并且将销售Pixar处理机和其他的高质量图形技术。然后还有我们的公司，名字叫做The Droid Works（机器人工场），销售EditDroid、SoundDroid以及制片人的其他计算机工具。所以计算机分部正在一点点地剥离出去，从研究阶段走向开发阶段，把系统放到真正的电影制作人手中。在此过程中人们似乎都有点晕乎乎的了。这就是我们当前的状况。

我是怎么进来的？好的，我们来看一下。我出生在南加利福尼亚的一个海军陆战队基地里，不到一岁就搬到了新泽西，很早就对计算机技术入迷了。从我家往山上走，就是莫雷山的贝尔实验室。

采访者：*那里就是你长大的地方？*

霍利：是的，我在纽约郊区的一个小地方新普罗维登斯长大，那里没有电影院，没有酒吧。但是，贝尔实验室就在山上，并且我在十五六岁刚能拿到工作许可时，就在那里的语言学部门里找到了一份计算机方面的工作。我无师

自通地学起了计算机——里面有些挺不错的研究项目。有一个研究员造了一台数字合成器，摆弄起来特别有意思。

从高中到大学，贝尔实验室的很多人都认识了我，他们会跟我说："嘿，过来，干吗不试试这个？"然后我就会照他们说的去做。断断续续地有好几年，我在一个认知心理学部门里工作，对于通信问题，尤其是人机通信的问题，做了些很有趣的基础研究。心理学家对计算机特别感到兴奋，因为它就像一个大镜头，你可以拿它来放大检查人们的思维中有些什么。

采访者：计算机是怎么帮助心理学家检查人们的思维的？

霍利：当你把计算机放到一个通信过程中去之后，信息就不得不通过这个狭窄的渠道，你可以用计算机对信息以全新的方式进行计数、处理和观察。计算机约束了用户，并且强迫设计师深入思考人们究竟想怎样对待一个应用，以及任务该如何最好地展现出来。它们帮助我们专注在通信问题上，强迫我们以从前不可能的方式领会到心理问题的边界。作为一种新的实验工具，计算机非常受重视。新的发现需要新的工具。



在高中和我的贝尔实验室经历之后，我去了耶鲁，修了音乐和计算机科学的双学位。我大部分时间都花在弹钢琴上，那有时候比摆弄计算机更有趣。不过，跟计算机科学相比，靠钢琴谋生可是非常痛苦的……

采访者：有很多人既搞音乐又搞计算机。你在两者之间看到什么相似之处吗？

霍利：那是当然。总的来说，我认为把计算机和艺术媒介放到一起，会让人对艺术和技术都感到耳目一新。

整个耶鲁期间，包括暑假，我继续待在贝尔实验室里。毕业后，我花150美元买了辆1966年的福特旅行车，一直开到了阿拉斯加，在那里划独木舟玩了一段时间。然后我回来，又在贝尔实验室工作了一段时间，然后去巴里·维科（Barry Vercoe）[1]在麻省理工的实验音乐工作室里待了一个月。它现在是那里的艺术和传媒技术中心的一部分。然后我去了巴黎的IRCAM。

采访者：你在IRCAM做了些什么？

霍利：我在那里弹了最最糟糕的钢琴曲。实际上，我是作为贝尔实验室的访问研究员去IRCAM的，目的是开发计算机音乐应用的用户界面原型。那是

① 计算机科学家和作曲家，著名的麻省理工学院媒体实验室创始人之一。

1983年秋天，人们刚开始在现实世界里使用位图图形显示器。在那之前，图形显示器只在研究机构里使用，比如施乐PARC。所以我在去巴黎的飞机上带了一台很大的贝尔实验室位图图形显示终端。我到了那儿，我也认为我带了所有的许可证明。至少，我准备好了IRCAM告诉我需要带的所有证明文件，但海关不这么认为。我想从门里穿出去，并且声明："我没什么要申报的，我没有。"但他们抓住了我。这只是一连串审讯和罚款的开始。结果是，他们没收了我的终端。由于法国一贯的官僚程序，约5周后我才拿了回来。我在巴黎的时间从一开始就烧掉了三分之一。

我发现自己有5周的时间无事可做。我把时间花在了学习IRCAM的计算机研究的进展上，并且专心研究了有关法语、法国食物和法国葡萄酒的知识。

采访者：*在无事可做时，巴黎可不是个坏地方。*

霍利：是的！我也没法拒绝在钢琴音乐会中演奏的邀请。一个疯狂的罗马尼亚钢琴师有天晚上走进我的办公室，说："嘿，想在音乐会里演出吗？"这个音乐会是在十一月我的生日当天，报酬也很有吸引力。除此之外，我在那时没法编程序，因为终端还在海关那里。音乐会的举办地点是蓬皮杜国家文化艺术中心的演出大厅，是一首全新巨作的首次公演，作曲家也会飞过来。听起来很不错，所以我说："当然，我的天啊！"



采访者：*你演奏了什么？*

霍利：那是一首为双钢琴加电子音乐带所作的奏鸣曲，一件庞大而稀奇的作品，有大概3万个相当随机的音符，会常常从两架漂亮的汉堡产D型音乐会大钢琴[①]上以刺耳的方式飞出来。

我们出演了音乐会。我相信在公众的眼里这场音乐会失败了，但对我而言这是场狂欢。半数的观众离场，其他很多人在喝倒彩。我的一小群IRCAM的朋友也在那里，大声喝彩，还给我献了花。这场演奏有一个小时长——这还是我们砍掉了一半慢速段落后的结果。对于非音乐家来说，这个乐谱像是五线谱纸落到一个重感冒的矿工手里的后果。不过，IRCAM的经历真是特别，那是一个非常美丽和有趣的地方。

采访者：*你是怎么从IRCAM跑到卢卡斯影业的？*

① 斯坦威钢琴的最大三角钢琴型号，为世界各地音乐厅所常用。

霍利：在IRCAM我碰到一个顾问，在他工作过的地方里就有卢卡斯影业。他帮我联系上了安迪·摩尔。在那时，AT&T正要和贝尔实验室分离，所以卢卡斯影业看起来更有前途。

于是我搬到西部。工作看起来很有趣；他们希望我为音频和信号处理器开发图形界面，还要想着音乐的东西。当然我答应了。工作面试本身就已经足够娱乐了……

采访者：卢卡斯影业的面试怎么会是娱乐呢？

霍利：在我们交谈了一会儿之后，安迪说："嗯，让我带你走一圈，看看我们正在做的东西。"那时，他们正在制作《夺宝奇兵2：魔域奇兵》，而我对此一点都不了解。每个人都咕哝着"夺宝2"，有一分钟我一直在想："很好，又一部汽车追逐的电影。"我们向混录棚走去，安迪解释着所有混录的问题，比如在混录了几周后，导演说了一句"嗯，我们要在这里砍掉15帧"，你就不得不从头再来。这就像让一个本科生在老式的打字机上重打他的学期论文一样。就这样，我们走进了混录棚。我感觉有点眩晕，因为我以前从来没去过加利福尼亚，而大屏幕上有一个戴着包头巾的可怜家伙……你看过这部电影吗？

采访者：没有。

霍利：嗯，那对你就有点剧透了。有一个家伙戴着包头巾，围着缠腰带，被锁在什么东西上，尖声惨叫着，一个坏蛋把手伸进他的胸膛，直接把他的心挖了出来。天啊，我刚下飞机不久，跑进一个奇怪的剧场，正好看到一个可怜的笨蛋被人挖了心。似乎这还不够糟糕，我看着混录师把这幕场景颠来倒去放了十几遍，试图要让嗤喽的声音听起来没问题。所以，就这样，这差不多就是我倾心于电影行业的过程了。我现在在这里已经差不多工作一年半了。

我正计划在1986年秋天回去读研究生。我一直觉得离开耶鲁后我只是放了一年的长假而已。有些更一般的东西我需要学习，但我没有时间，特别是现在我们的开发任务越来越重了。

采访者：你要回学校学习什么？

霍利：主要是计算机科学和音乐。如果计算机部门仍然是完整的话，我希望一直做现在这儿做的事情。比如，把计算机化的音符、声轨和计算机产生的



电影组合到一起。但是，很快这儿就不再可能这么做了。其中一个原因是，图形和音频部门将各自成立独立的公司——这真是可惜。图片和声音过去合作得非常好，看起来它们本该就在一起。从经济角度看，我想把计算机部分拆开是有效的，至少接下来几年是这样。

采访者：难道彼此之间没有沟通渠道吗？就算两部门都被拆成独立公司，它们就不能继续合作了吗？

霍利：当然，在最好的情况下可以是这样。但无论如何不可能像现在这样，使用同一幢办公楼，大家可以彼此来回串门看对方的机器。搞研究的决策者们还是保持着联系，他们讨论着彼此的梦想和要做的事情，但是，要实现绝非易事，短期内至少不会。

采访者：你对在卢卡斯影业的编程工作有什么特别喜欢的地方吗？



霍利：卢卡斯影业看起来是个计算机科学研究的理想场所。这儿我们有着这个异常丰富的通信载体——电影——又有图像又有声音，又有对话又有故事，又有音乐又有特效。当你用信息技术来展现这些东西，比如说计算机编程时，所有的新发现都会散架。屏幕上的颜色是如此丰富，场景是那么引人入胜，这对计算机是个巨大的负担。我们的想法不是复制你可以出去拍摄的东西，而是使用计算机把图像和声音合成到一起，要有足够的复杂度，使看电影的人觉得效果漂亮且有趣。使用当前仿佛出自青铜器时代的计算机，计算这些东西需要花很长时间，但这可能是值得的。

采访者：你喜欢计算机和编程的什么方面？

霍利：我还在试图搞清楚。我去年写了4万行代码。在多年的受挫之后，人们会对写蹩脚的计算机程序感到厌倦。我喜欢编程的地方，则是它可以真正帮助你思考我们该如何沟通、如何思考，逻辑如何运作，创造性艺术如何产生。计算机是沟通和信息的工具，而沟通是件美好的事情。这就是电话和个人计算机这么快就对人们的生活产生影响的原因。计算机可能是看待沟通问题的终极工具。

我喜欢把音乐和计算机组合到一起这个想法，因为音乐看起来是一种特别丰富的载体。音乐也和感情密切相关。跟图片相比，在音乐上你可以用更少的计算量产生更多的感情反应。对作曲家而言，现在还没有什么好的字处

理器；当然，音乐的结构确实要求严苛。乐谱很复杂，而时间，或者说实时性，是其中的关键要素。要让计算机实时演奏音乐非常困难。举个例子，想象一下你给表演者伴奏要做些什么就知道了。你必须完全理解伴奏者是如何通过乐谱跟住表演者的。你必须记住乐谱，必须把空气中的颤动转变成音调，听准音符，利用模式匹配确保和演出保持同步。这个问题非常困难，但不久后建造有趣的计算机伴奏器的技术就会成熟，那和现在电子琴上的“Samba”节奏开关会大不相同。用我们现在的设备已经差不多可以做到了。

采访者：你是否认为计算机会影响艺术？

霍利：会有深远的影响。就像计算机影响了我们日常生活的方方面面一样，它对艺术也会有同样全面的影响。

很多新的东西可以用计算机完成。我认为，好点子往往是新瓶装旧酒产生的。如果说计算机有什么做得特别好的，那就是它可以处理海量的信息，将其混合到一起，并让你摆弄产生的结果。这会帮助催生新的点子。



困难的、有点类似哲学的问题仍需单独解决。举个例子，在字处理器里写文章和在打字机上写文章大不相同。当使用打字机时，你需要思维连贯，写出有重点的语句，并确认表达出了你想要传达的信息。你受到了约束，因为你不想重打一遍这见鬼的东西。说话也很相似：你讲话的时候也没法回过去修正小语法错误。打字或打电话时，你得花更多的时间计划讲什么，以及确保清楚地表达了想要说的意思，这样的话，你就不需要重新打字或是重复要说的话。用字处理器时作修改则容易得多：你可以跳来跳去，忽前忽后，从任何地方剪切-复制-粘贴-抓取。这样，写作时思路中断一两下也就变得很容易发生。现在，英语老师都在抱怨他们可以分辨出学生的作业是用字处理器还是打字机完成的，因为字处理器写出的文章看起来就像一盘大杂烩。这个说法还是有道理的。计算机是新技术，它会影响人们的创作过程。你可以用它做些不一样的事情，但你得意识到你在以不同的方式创作。

就像我前面说过的，如果说计算机擅长什么，那就是它可以帮你组合各种不同的元素，就像厨子可以用同样的原料做出新的菜品一样。人们将不得不去适应耗费时间以不一样的方式来组合事物。

采访者：在你看来，对声音和电影制作用计算机程序进行操作产生了哪些影响？

霍利：字处理的类比在电影制作方面也正好适用。今天，在电影里混合声效

时，你坐在混录棚里一张大桌子前，看着电影在屏幕上滚动。你调节旋钮，注意观察着峰值计。等到停下来时，你就混录好了一段。你已经过了一遍，所以你得倒带才能再听一下。通常，在你等待倒带时有5分钟啥都干不了。在过去，这段时间被用来规划下一遍混录时使用的手势和要改善的地方。但当用上计算机后，这段时间就没有了。

有次，我坐在一部叫*Latino*的电影的混录棚里，他们缺少了一个声效。一个战士从草地上爬过去，你可以听到他手臂伸向雷管时的沙沙声。就在他要引爆炸弹前，他被子弹击中，翻倒在地。身体翻倒的声音缺了。混录师想做的是，把手臂磨擦的沙沙声复制下来，放到后面一些，可能要做些调整，音量均衡器向上推到一个不同的位置，然后当作身体翻倒的声音用。做这些需要至少5分钟，因为他需要按一个按钮，在话筒里说："嗨，乔，你能不能把第三声轨上的福莱（Foley）音效[1]磁带卷先错位一下，往前进上175帧？"两分钟之后，有声音说："好了。"于是混录师试了一下磁带，当然磁带的位置仍然不很正确，所以他们还得继续前后调整，直到找到正确的位置。然后他们把声音复制下来，再把磁带重新同步到原来的位置，5分钟就过去了。使用我们现有的技术，你只需轻点计算机的屏幕一两次，不但复制已经完成，你已经开始试放了。只要几秒钟就行。你不再需要等待回绕磁带。这就很不一样了。计算机仿佛一直说："我已经好了，当然。继续吧，任何时候你准备好了，我就准备好了！"



一天，我正在和弗朗西斯·科波拉（Francis Ford Coppola）[2]的混录师聊天，他过来看我们的机器。这是个灵巧又淘气的家伙，正被技术搞得晕乎乎的。谈到现在有那么多新工具可用，有那么多新东西需要思考时，没想到他说："不要忘记这5分钟的倒带时间从来就没有被浪费。如果你是个好的混录师，你总是在计划下面你要做的手势和效果，你总是为了有5分钟的连贯操作而在脑海里练习这一过程。使用机器，你就丧失了这一思考时间。"你得到了一些东西，但你也丧失了一些东西。每得到一个新特性，看来总有一些想要的老特性会消失，或至少被扔在了一边。但是，还是有空间让两者共存的。人们必须意识到，出品好的艺术、好的电影、好的音乐仍然是需要花时间的。

① 需要与荧幕同步的特殊音效，名称源自音效师Jack Foley。

② 著名美国导演，最有名的作品是《教父》（*The Godfather*）三部曲、《现代启示录》（*Apocalypse Now*）和《吸血惊情四百年》（*Dracula*）。

音乐也有同样的问题。合成器和电子鼓正在被广泛使用，而现在还可以用计算机来对其进行低成本的操控。流行风潮经常是盲目奔向新技术，而你一定要意识到在这个过程中放弃了什么。

采访者：给我讲讲你在卢卡斯影业完成的音乐程序方面的工作吧。

霍利：上个夏天我带了个朋友来，他是康奈尔计算机科学系的博士生。我想，我们可以在MusicDroid上做开发。MusicDroid的目标是成为音乐家的工具，终极的计算机音乐家工具。我们希望能拿出一套系统让约翰·威廉姆斯（John Williams）[①]用来尝试交响乐谱，可以用一堆合成器组成的交响乐团，而不是租借伦敦爱乐乐团，来进行试验，然后很容易地把乐谱打印出来。另外，我们也想让他可以专门为电子乐器谱写音乐，用从来没人想到过的方式。这就是我们的愿景。

我下面要展示的程序是那个方向上的第一小步。这儿有个便宜的合成器，是一千块买来的现成产品，它有很多不同的音色……我们发出的风琴声真的很不错。[霍利边说边开始在合成器上演奏。]

实际上，如果我把这个开起来……我有一个一百块钱、超级便宜的哈蒙（Hammond）电风琴回响器可以完成这个把戏。立刻感觉来到圣约翰大教堂[②]了吧？这些合成器非常便宜，而你现在还可以直接从计算机上控制这些东西。比如，我想这已经可以工作了。多半听起来很恐怖，但你永远不知道你会发现些什么……

采访者：这是你写的？

霍利：是的。[霍利不停地摆弄着程序和合成器。]它可以以很快的速度演奏乐谱，[说到这里，他演示了一下]也可以上下颠倒演奏——那样的结果挺有趣的。倒过来之后，低音跑到了上面，而高音跑到了下面；大调变成小调，而小调变成了大调。你会得到一些非常不同的东西，充满着奇异的美。做这个非常简单。5行计算机程序就可以把音乐倒过来，产生全新的东西。

现在我们看到的是乐谱编辑器里显示的我刚才在键盘上演奏的音乐。你可以用这根滚动条移过去看作品尾部。另外，记得刚才我在结尾处弹的那几个笨拙的和弦吗？你可以在那部分放大缩小。如果需要的话，你可以放大到

① 美国音乐家，曾为包括《星球大战》在内的多部影片配乐。
② Cathedral of Saint John the Divine，纽约曼哈顿著名的大教堂。



最底下进行摆弄，那可能非常有用。

你可以看到，音乐从哪里来没有关系，关键是怎么来。我们在进行组合，有老原料—— 一首美丽的巴赫序曲；有新方法——计算机控制的合成器；外加实现完全和声翻转的简单程序。

在某种意义上，巴赫在世时也做了类似的事情，但他的本事在于，他了解他那个时代关于音乐的一切。他是他那个年代所有乐器的大师，熟知所有重要的乐器风格和作曲流派。他能将不同的风格——富丽堂皇的法国宫廷风格、德意志北方的巴洛克风格、意大利风格——全部揉合到一起创作出新的乐曲。

采访者：除了写这些程序之外，你帮助进行设计吗？

霍利：是的。我在这儿的角色是开发一些底层接口的软件，像图形库，触摸屏和合成器的设备驱动程序，诸如此类。但我在设计问题上也很有发言权。



我在一些有趣的项目上作过研究和设计工作。这儿有个我开发的小巧的字处理程序。我有一个叫Books的目录，里面放了一些伟大的文学作品，比如《爱丽丝漫游奇境》、《白鲸记》、《战争与和平》……

采访者：你自己打字进去的吗？

霍利：不，没有。你寄一张35英镑的支票给牛津文本档案库，他们就会给你寄一盘装满材料的计算机磁带。我对在数据库中使用它很感兴趣。

我把所有这些文本搞进了一个大的词汇索引，一个类似大辞典的东西。我可以在这个字处理器程序里打字，然后当我想找一下某本书或某个作者是怎么使用某个单词时，只要用鼠标点一下，然后，呼拉，就会弹出一个小窗口，里面有五六条例子，展示数据库里那些伟大的作家是如何使用这个词的。我们的词汇模式倾向于每次走同样的路线，但现在突然间，伙计，我可以看到托尔斯泰和其他的作家是怎么使用我刚才键入的那个单词的。

另一个程序接受文本，然后，使用同样的数据库，会胡乱写出随机的基本符合语法的英语来完成句子。如果你打字时显示了作者块，点一下按钮，这个程序就会从你写的最后一个单词开始，朝着随机的方向发射。每隔一会儿，你就会有些意外发现，会找到一个你原先想不到的点子。这个程序实际上是这儿SoundDroid部门的一个研究项目，研究大型数据库的管理问题。

采访者：在你追求的领域里，有什么人对你有特别影响吗？

霍利：当然。所有这儿的人，安迪·摩尔、阿尔维·史密斯、埃德·卡特穆尔，还有贝尔实验室的汤姆·兰道尔（Tom Landauer）和麦克斯·马修（Max Matthews）。马文·明斯基和艾伦·凯的观点也时常启发我的灵感。我不认为有哪一个人影响最大，因为一般而言我对综合各种技术更感兴趣，而不是深入研究某个狭小的领域。

采访者：*你看到计算机有什么问题吗？*

霍利：嗯，主要是些“普遍”的问题，所有新技术都会烦恼的问题，就是人们认为他们可以不付出任何代价就得到新东西。营销人员把计算机说成是省时利器，宣称你有了计算机后生活会变得简单得多。这听起来诱人，但却不着调。我决不会暗示说，在屏幕上快速翻动《爱丽丝漫游奇境》，就可以代替坐在火炉跟前拿着漂亮的精装书，望向英格兰某处的鲜花走廊。这不是计算机要做的事情，当我看到人们急着加入新行列、抛弃高度发展的老技术时，我就感到很伤心。这是一个大问题。

采访者：*你不是看到音乐中发生的事情了吗？举一个例子，古典乐器。你是否认为它们最终会被合成器所取代？*



霍利：既是也不是。有计算机的目的不是要取代什么、模仿什么，或者愚弄别人相信他们是在听那些古旧的美妙技术。如我前面所说，你不应该使用计算机去创作一张你本应该到外面拍来的图片，或者去弄出风琴的声音。我们需要的是能够达到和理解让图片或者声音美丽动人的复杂度。模拟是理解问题的好方法，但模拟不是目的。我更有兴趣做我以前没法做的事情。比如，我以前没法做的一件事是同时演奏教堂管风琴和木琴，并且就在办公室里把声音颠来倒去。今天，用以前一台钢琴的价格，消费者就可以买到能用计算机控制的合成器，可以产生的声音具有非常漂亮的复杂度。将来会有一天，你只要按下一个按钮，计算机就可以胡乱编出一段背景音乐来，只要你希望它这样做。每一种强大的科技都有阴暗面。

我一个月前回了趟耶鲁，去看望了我的老钢琴教师，他已经七十多岁了。他提到他去参加某个学院的老友晚宴，一个怪老头站起来讲了一段话……一个不断重现的主题……无论何时当你拥抱新事物时，老事物也同时丢失了。在那时，葛洛里亚飓风[1]正穿过美国，人们抱怨他们没有电灯了。在我的老

① 1985年袭击美国的飓风。

师是耶鲁学生时，他们没有电灯，只有蜡烛。他说，情况没那么糟糕，不仅飓风不会影响蜡烛的使用，蜡烛也没有荧光灯的嗞嗞声。蜡烛照亮书本的方式都不一样。

愿意呆在黑暗里、继续无知的人们永远不会了解烛光下读书是何种滋味，也不会了解在漂亮的音乐厅里演奏老鲁特琴是什么样的感觉。但如果人们勇于学习的话，他们除了探索新科技外，也会研究老科技，这样他们才能同时重视两者、理解大图景里各个事物的关系。人们可能会很粗浅地使用计算机技术，而不去充分重视技术的背景和来源，这样的风险很大。不过，我还是有些信心，人们或多或少总会去做些正确的事情。但有时候我还是会感到惊慌。

世界总是在改变。巴赫活在300年前。300年并不算长，在巴赫和我之间数不出几个爷爷。但是他的日常生活是那么不一样。他用羽毛笔把音乐手写在纸上。他坐不了飞机，没法飞跃整个国家到麻省理工去找几个人聊天，但我可以。他的生活圈子超不出同一个国家那几个大同小异的小镇，周围也都是差不多同样的人。看看他写出了什么。当然，这种比较不太公平，巴赫这样的人不是每个世纪都有的。但是，通信技术和运输技术改变了我们与他人打交道的方式，不是每种改变都是我们喜欢的。看一下社会里人与人的关系，这和一些年前已经不一样了。某些方面更好，某些方面更糟。今天，人们认为，如果不喜欢在曼哈顿的工作或者感情生活，可以搬到加利福尼亚重新开始，这是理所应当的。或者你完全可以跳上飞机，去南太平洋的小岛上清理一下大脑。改变道路是相对容易的选择，这样人们也就不再重视解决问题的需要。你只需按一下按钮，音乐就变了；拨一下开关，灯就灭了；跳上飞机，你就到了挪威。这是一个不一样的世界。有些东西变得更好，有些变得更糟。当新科技带来的激动平息时，我们也就可以开始通盘考虑、公平看待这两个方面了。



续写传奇人生

当时在Droid Works开发SoundDroid软件，后来去了NeXT工作，在此期间开发了世界上第一个数字图书馆，并创造了莎士比亚和其他经典著作的数字版本。1993年到2002年间，霍利在麻省理工学院任教，并成为MIT媒体实验室特别项目负责人，其研究横跨多个领域，包括心理学、电脑音乐、数字

视频编辑、人机界面以及纪录片摄影等。

霍利是多个研究项目及组织的创始人或联合创始人，主要包括MIT的GO探险项目、Things That Think（探索数字媒体注入日常物品的无限方法）、Toys of Tomorrow（联合多家世界领先的玩具公司发明好玩的东西）、Counter Intelligence（探索家用技术）以及Friendly Planet（致力于发展中国家儿童教育的非营利性公司）等。他还曾是1998年美国珠峰探险的技术负责人。

霍利同时还是一位钢琴家和风琴演奏家。2002年，他在第三届范克莱本国际杰出业余钢琴家大赛中获得冠军；他曾以独奏或与管弦乐队合奏等形式多次演出；2006年，他还曾与马友友一起合作演奏过《婚礼进行曲》。



词 汇 表

8080：英特尔公司制造的一款微处理器芯片。许多早期的微型计算机都采用这款芯片。

ALGOL：算法语言（ALGOrithmic Language）。一种用于表达和控制算术及逻辑过程的国际编程语言。

算法（ALGORITHM）：以有限的步骤解决问题的一整套规则或过程。

ALTAIR计算机（ALTAIR）：MITS公司生产的早期微型计算机，成套销售。



ALTO：施乐公司帕洛阿尔托研究中心开发的工作站计算机。Smalltalk高级编程语言、叫做鼠标的输入设备以及用于连接各台ALTO计算机的网络互连技术，是ALTO的特色。

APL：编程语言（A Programming Language）。一种面向算法/过程的高级语言。一种通常需要用到特殊终端的交互式语言，在数学和科学工作中用处很大。

应用程序（APPLICATION）：为完成特定用户任务（如工资单）而编写的程序，有别于通用或实用程序。

人工智能（ARTIFICIAL INTELLIGENCE）：指某一类机器的开发，这类机器能够执行通常与人类智力（如学习、自适应、推理和自动自校正等）有关的功能。

汇编器（ASSEMBLER）：一种将低级计算机源代码转换成可执行机器码的程序。它还常作为汇编语言（assembly language）的同义词。

汇编语言（ASSEMBLY LANGUAGE）：低级计算机语言，基于机器语言对应的助记符语句产生的源代码，其中每个语句与实际机器指令一一对应。

后台处理（BACKGROUND PROCESSING）：在高优先级或快速响应程序处于不活跃状态时，执行低优先级作业。

BASIC语言（BASIC）：适用于初学者的多功能符号指令码（Beginner's All-purpose Symbolic Instruction Code）。一种常见的高级计算机编程语言，由达特茅斯学院开发。

β测试（BETA TEST）：在实际使用环境中，由开发商之外的人员执行软件或硬件的测试。

位（BIT）：计算机所能度量或探测的最小量，相当于二进制数字0或1。8位组成1字节。

位图（BIT-MAPPED）：指的是计算机显示系统，其中屏幕上的每个像素对应于计算机内存中的一个（黑白）或多个（彩色）位。

布尔代数（BOOLEAN ALGEBRA）：一个逻辑函数系统，以英国数学家乔治·布尔（George Boole）的名字命名，使用AND、OR、NOT、EXCEPT、IF和THEN等运算符推导逻辑问题的答案，其中每个元素可以具有两种状态之一，一般为逻辑真或逻辑假。



字节（BYTE）：计算机内存和磁盘存储的度量单位。1字节包含8位，可以存储一个字符（字母、数字或标点符号）。

C语言（C）：贝尔实验室的Dennis Ritchie开发的编程语言，设计目标是优化运行时间、规模和效率。C语言不依赖于特定机器，因此C程序可以在不同种类的机器之间自由迁移，一般不用修改就能正确运行。

CAD：计算机辅助设计（Computer-Aided Design）。自动化设计和制图系统。

只读光盘（CD-ROM）：Compact Disc, Read-Only Memory的简写。一种光学存储系统，利用激光来探测旋转盘片表面的凹点，可存储多达540兆字节的信息。

CEO：首席执行官（Chief Executive Officer）。

COBOL：面向商业的通用语言（COmmon Business Oriented Language）。一种使用英文语句的编程语言。

代码、编码（CODE）：特定编程语言的字符语法和规则系统。当程序员“编码”（code）时，表示他们正在使用特定的编程语言编写计算机程序。“看代码”（looking at code）一般指阅读程序清单。

COMDEX：计算机经销商展览会（COMputer Dealer’s eXpo），计算机硬件、软件和相关产品的年度展会。[①]

编译（COMPILE）：将高级语言编写的程序“挤压”成机器更容易解释的程序，也即这个程序执行速度会更快。

编译器（COMPILER）：一种计算机程序，将程序员用高级语言编写的程序转换成可执行的机器码。

计算机图形（COMPUTER GRAPHICS）：用显示器和打印呈现计算机生成的图表以及相关艺术作品。

CONDOR：Condor计算机公司（Condor Computer Corporation）开发的数据库管理系统。



CPU：中央处理单元（Central Processing Unit）。微型计算机的一部分，负责解释和执行指令的控制电路。

CP/M：微处理器控制程序（Control Program for Microprocessors）。数字研究公司开发的8位微型计算机操作系统。

交叉编译（CROSS-COMPILE）：使用一个叫做交叉编译器的特殊编译器，在一种计算机上编译程序，生成可以在另一种计算机上运行的可执行代码。

光标（CURSOR）：视频终端上的指示符，用来突出待修正的字符或准备输入数据的位置。

数据库（DATABASE）：一种电子文件，包含关联条目、引用或对主题的摘要。

数据库管理系统（DATABASE MANAGEMENT SYSTEM）：一个软件系统，主要功能是允许用户创建、操作和检索数据库记录，进行处理和显示。

数据结构：以存取为目的，用来组织一组数据或信息的方法，如文件、

① 1979年至2003年，每年11月在内华达州拉斯维加斯举办的计算机展会。

字符串或矩阵。

调试（DEBUG）：查找、修正或排除计算机程序中的错误。

数字化（DIGITIZE）：将模拟量转换成用二进制或以2为底的数字表示的数。

Dynabook：假想的强大计算机，小到能装进书包。Dynabook的概念源自艾伦·凯。

编辑器（EDITOR）：用来编写源代码的通用文本编辑程序。

电子邮件（ELECTRONIC MAIL）：允许用户利用电子计算机创建、发送和接收信息的系统。

专家系统（EXPERT SYSTEM）：一种计算机系统，可模拟决策过程，一般与人类思维相关。

可扩展（EXTENSIBLE）：可扩展语言允许用户定义新的元素或修改既有元素。



第五代计算机（FIFTH GENERATION）：这个词描述的是目前正在开发的一种计算机系统，该系统极度依赖处理速度和人工智能。

文件服务器（FILE SERVER）：一个计算机网络中，专门存储和处理网络文件的计算机和软件。

固件（FIRMWARE）：存储在计算机的永久存储器或ROM中的程序。

字体（FONT）：一整套同样尺寸和风格的字母、数字和标点符号等。

Forth语言（FORTH）：一种微型计算机上用来解决一系列问题的可扩展编程语言。

FORTRAN语言（FORTRAN）：公式翻译（FORmula TRANslation）语言。一种专为数值计算开发的高级编程语言。

功能规格说明书（FUNCTIONAL SPECIFICATION）：描述语言或系统的操作特征和限制。

GEM：图形环境管理器（Graphical Environment Manager），Digital Research公司开发的图形用户界面，使用图标表示文件，窗口表示目录和子目录，菜单用于用户控制。

千兆字节（GIGABYTE）：度量单位，约等于十亿字节。

hack：俚语，指编写计算机程序。

黑客（HACKER）：俚语，指程序员，尤指在计算机科学、编程和设计等方面技艺高超的程序员。

大力神图形卡（HERCULES CARD）：大力神计算机技术公司（Hercules Computer Technology）制造的一种显示适配器卡，可以显示高分辨率的文字。

高级语言（HIGH-LEVEL LANGUAGE）：面向问题或过程的语言，不同于面向机器或助记符的语言。

交互（INTERACTIVE）：指计算机及其用户之间的双向通信，涉及用户的命令和响应。

接口（INTERFACE）：两种特性不同的部件之间的交界，或者两种组件、电路、设备或系统要素之间的互连。



解释器/解释型语言（INTERPRETER/INTERPRETIVE LANGUAGE）：解释器指的是运行期间将源代码翻译成机器语言的程序。

LISA计算机（LISA）：苹果计算机公司制造的微型计算机。

LISP语言（LISP）：表处理语言（LISt Processing）。一种操作符号串和递归数据的解释型语言。

表处理（LIST PROCESSING）：处理有序序列（list）中存储的数据。

局域网（LOCAL AREA NETWORK）：一个系统，使小范围内的大量计算机、外设或终端得以共享资源。

LSI：大规模集成电路（Large Scale Integration）。一种计算机芯片制造技术，可以在一颗芯片内容纳成千上万个逻辑门。

邮件合并（MAILMERGE）：许多字处理程序提供的一项功能，用于创建个性化套用信函。

大型机（MAINFRAME）：一个大型中央计算机系统。大型机一般字长32位，内存容量从512KB到16MB不等。[①]

① 注意本书写作时间为1986年。

兆字节（MEGABYTE）：1 048 576字节。

微型计算机（MICROCOMPUTER）：一类廉价的计算机系统，通常由单颗微处理器芯片和支持组件（包括半导体内存）组成，一般装在小机箱里，附带一个键盘。

微处理器（MICROPROCESSOR）：单芯片上的完整处理器，用作微型计算机的CPU。

Microsoft Word：微软公司开发的字处理软件。

小型计算机（MINICOMPUTER）：一个中型计算机系统，通常采用半导体或磁芯存储器，并提供4K到64K字的存储空间，时钟周期为0.2到8微秒或更短。

MIS：管理信息系统（Management Information System）。一个记录和处理业务数据的系统。

Modula-2：Volition系统公司设计的编程语言。特性包括：模块、分离编译、程序库、并发处理和低层机器访问等。

模块（MODULE）：在硬件中，模块指可互换的“插件”项，可以结合其他可互换项，形成一个完整的单元。在软件中，模块指执行特定任务的指令序列。

MS-DOS：微软磁盘操作系统（MicroSoft Disk Operating System）。微软公司开发的16位微型计算机操作系统。

MultiMate：SoftWord系统公司开发的字处理软件。

多进程操作系统（MULTIPLE-PROCESS OPERATING SYSTEM）：一种操作系统，允许用户一次建立两个或更多进程实例。

八进制（OCTAL ABSOLUTE）：以8为底的进位制，而不是二进制以2为底和十进制以10为底。

OEM：原始设备制造商（Original Equipment Manufacturer）。为其他制造商提供设备或零部件的制造商。

操作系统（OPERATING SYSTEM）：一种程序，负责管理和控制计算机系统内部的处理过程。



解析（PARSE）：将编程语句分离成可被翻译成机器指令的基本单元。

PASCAL语言（PASCAL）：一种高级编程语言，为纪念法国数学家和哲学家布莱兹·帕斯卡（Blaise Pascal）而命名。

PC-DOS：IBM PC上使用的一个MS-DOS操作系统版本。

PDP小型机系列（PDP SERIES）：数字设备公司生产的一系列小型机，例如PDP-8、PDP-9等。

PL-1：Programming Language 1的简写。专为IBM计算机打造的编译器。Digital Research公司开发了一个微处理器用的版本。

原语（PRIMITIVE）：基础或基本单元。通常指的是最底层的机器指令。

编程语言（PROGRAMMING LANGUAGE）：一种人工语言，例如BASIC和PASCAL，提供用来编写计算机程序的特定语法、规则、字词、短语。



公共领域（PUBLIC DOMAIN）：指代无版权软件，公众使用时通常无需支付任何费用。

R:BASE系列数据库（R:BASE SERIES）：Microrim公司开发的单用户关系数据库程序。

实时（REAL TIME）：在规定时间内快速及时地给出问题的答案。

递归（RECURSION）：同一个或同一组操作的重复执行。

检索（RETRIEVE）：在包含大量条目的存储系统中，定位和获取一个或一组指定条目的操作。

ROM：只读存储器（Read-Only Memory）。计算机的固定存储器。

扫描器（SCANNER）：一种仪器，检查或采样不同进程的状态、文件或者物理状态。

屏幕接口（SCREEN INTERFACE）：计算机程序中控制哪些内容出现在显示屏上的组成部分。

半导体（SEMICONDUCTOR）：导电性介于金属导体和绝缘体之间的材料（通常是掺入其他材料的硅），具有特殊性质，可以控制和使用通过它的电流。

SNOBOL：面向字符串的符号语言（StriNg-Oriented symBOlic Language）。一种操作字符串的编程语言。

电子表格（SPREADSHEET）：一种软件包，利用电子计算机模拟商业或科学的计算表。

Switcher程序（SWITCHER）：安迪·赫兹菲尔德编写的Apple Macintosh实用程序，允许用户在内存中一次存有多个应用程序，并在它们之间进行快速切换。

系统设计（SYSTEMS DESIGN）：描述系统各部分之间工作关系的规格说明。

分时系统（TIME-SHARING SYSTEM）：一种计算机系统，其中央计算机的时间由多个用户共享。

自顶向下编程（TOP-DOWN PROGRAMMING）：一种结构化编程方法，首先设计整个系统，然后分别处理一个个模块。

TRS-80：Radio Shack公司制造的微型计算机。

UNIX：贝尔实验室专为小型和微型计算机开发的实时操作系统。

用户友好（USER-FRIENDLY）：各种各样的人都很容易使用和理解的计算机程序。

用户界面（USER INTERFACE）：计算机程序中与用户交互的组成部分。一般指用户在显示屏上看到的那些内容。

VAX：数字研究公司制造的一种计算机。

风险资本（VENTURE CAPITAL）：投资初创企业的资本。

VisiCalc：Software Arts公司开发的商用电子表格程序。

VisiOn：VisiCorp公司开发的集成操作环境软件包，允许用户同时在屏幕上使用任意数量的应用程序。

VMS：虚拟内存存储（Virtual Memory Storage）系统。在该系统中，内存分为主存和辅存两部分，后者被划分成4096字节或更大的“页”。这个系统允许程序使用比现有主存容量更大的空间，因为必要时程序的部分模块可



以从主存转移到辅存（通常是磁盘驱动器或其他大容量存储设备）。

窗口（WINDOW）：CRT屏幕上的指定区域，可以显示不同于主显示区域的信息。

WordPerfect：Satellite软件国际公司开发的字处理程序。

字处理器（WORD PROCESSOR）：专为编排和编辑文本打造的程序或机器。

WordStar：Micropro国际公司开发的字处理软件。

自动换行（WORDWRAP）：字处理的一个附加功能，指单词在一行已满的情况下自动转到下一行。

工作站（WORKSTATION）：一种集字处理、数据处理和数据通信设备于一身的计算机。也指连接到更大的计算机的终端，该计算机专门执行某个或某些任务，如商业、工程或科学应用等。



XEROX PARC：施乐帕洛阿尔托研究中心（Xerox Palo Alto Research Center）。施乐公司于1970年设立的技术研究中心。

Z80：Zilog公司制造的8位微处理器。

附　　录

```
FILE:    PESO1    21-SEP-74 14:30:03      PAGE 30

1
2
3        ; PRPLC
4        ; PROPAGATE LOCAL CONNECTION:
5        ; PARAMS: A:LC, B:CN
6        ; IF CN=(DP,GN,APN), CALL DEFP(CN,DP,GN,APN)
7        ;  TO DEFINE THE PIN.
8        ; IF LC=(CP), SCAN BACKPANEL WIRELIST AND FOR EVERY
9        ; (CN,BWPN) FIND LC ON THAT CARD AND CALL PRPLC(LC,CN)
10       ;  RECURSIVELY.
11       ; IF LC=(ESCC,REL), SCAN LC'S AND CALL PRPLC(LC,CN) RECURSIVELY.
12       ;
13       PRPLC:  AOS     CLOCK           ; FOR DEBUGGING PURPOSES...
14               TRNN    A,777777        ; RETURN ON NULL LC
15               RET
16               TRNE    A,400000
17               JUMPA   PRPLC2  ; BRANCH IF ESCAPE BYTE
18               TRZN    A,200000        ; UNLESS CONNECTOR PIN
19               JUMPA   DEFP            ; CALL DEFP AND RETURN
20               SKIPGE  RGPRI(B)        ; RETURN IF CARD IS MASKED
21               RET
22               MOVEM   A,PRPLLC        ; SAVE IN CASE OF ERROR
23               MOVEM   B,PRPLCN        ; LIKEWISE
24       ; SCAN BACKPANEL WIRELIST:
25               IMULI   B,MXCP
26               ADD     B,A
27               MOVE    X,B
28               HX      X,RGGCH
29               TRZN    X,400000
30               JUMPA   [SKIPN  NOSLER
31                        ERR(PRP,<S-L BACKPANEL MISMATCH, LC: >)
32                        RET]
33       ; NEXT LOAD PIN IS IN X, FINISHED IF 0.
34       PRPLC1: JUMPE   X,[RET]
35               MOVE    B,X
36               IDIVI   B,MXCP  ; CN TO B, BWPN TO C.
37               MOVE    A,RGCT(B)       ; CARD TYPE TO A
38               MOVE    A,DNCTP(A)
39               PUSH    P,X             ; SAVE X FOR RECURSIVE CALL
40               MOVE    X,C             ; BWPN (CP) TO X
41               HXR     A,A,DNCPH
42       ; NOW LC AND CN ARE SET UP IN A AND B.
43               CALL    PRPLC
44               POP     P,X             ; RESTORE X
45               HX      X,RGGCH         ; FETCH CDR OF THE WIRE
46               TRNN    X,400000
47               JUMPA   PRPLC1
48               ERR(PRP,<GARBAGE WIRE, LC: >)
49
50       ; THESE VARIABLES CONTAIN THE LC AND CN PASSED TO PRPLC
51
52       LS(PRPLLC)
53       LS(PRPLCN)
54
55       ERRPRP: SAV(<A,B,C>)
56               MOVE    A,OJFN
57               MOVE    B,PRPLLC        ; GET LC
58               JNUM(4)
59               JTXT(< CN: >)
60               MOVE    B,PRPLCN        ; GET CN
61               JNUM(4)
62               REST(<C,B,A>)
63       ;
64       ; ESCAPE BYTE IS IN A
65       ; GET BASE FOR RELATIVE ADDRESS:
66       PRPLC2: MOVE    C,RGCT(B)
67               MOVE    C,DNCTP(C)      ; BASE
68               LDB     X,BPREL(A)      ; RELATIVE ADDRESS
69               LDB     D,BPESCC(A)     ; COUNT TO D
70       PRPLC3: JUMPLE  D,[RET]
71               PUSH    P,X
72               HXR     A,C,00
73               PUSH    P,B
74               PUSH    P,C
75               PUSH    P,D
76               CALL    PRPLC
77               POP     P,D
78               POP     P,C
79               POP     P,B
80               POP     P,X
81               ADDI    X,1             ; INCREMENT POINTER
```

这个将布线表编译成硬件模拟器的编译器展示了查尔斯·西蒙尼在1972年写的“早期匈牙利风格”代码。他采用当时流行的“Tenex”编码风格为PDP-10计算机编写了这个编译器，并最早使用了基于类型的命名法，该命名法对机器语言或高级语言代码同样适用。



November 1971

Format of the Pattern Match Main Stack PM

During pattern match (PM) 3 stacks are maintained: the main, S and P stacks. The main stack is a continuation of the interpreter stack. Its 4 word entries however do not have headers. The S and P stacks are implemented as lists in the list storage. A 4 word entry represents the state of a simple pattern which has already matched a substring of the left operand.

0	(X4):		f2	f1
1	(X5):	lenf	rib	(B1) sil	alt
2	(X7):		(B4) index	(A0) pos
3		return		

pos: pointer to the first character of the substring matched. (Note that the left operand is stored in STY format, 1 character/word in the stack.) If the matched substring is null, pos points to the character after it.

西蒙尼针对Snobol系统内部机理编写的文档片段。文档是在项目完成后着手编制的，清晰而准确。相比之下，后期实现跟前期设计文档往往会有出入。

```
177       % DUAL 4-INPUT GATE WITH TRUE AND FALSE OUTPUTS
178       % 8-7063          8-7097
179
180       DD        "8-7063+",0                    ;
181                           FAULT    LA,L,8      ;
182                           FAULT    HA,H,8      ;       % P8
183                           FAULT    LR,L,9      ;       % P9
184                 OR        $R                   ;
185                           FAULT    LS,L,11     ;       % P11
186                 OR        $S                   ;
187                           FAULT    LPO,L,12;           % P12
188                 NANDN     PO                   ;
189                 LDR       $A                   ;
190                 COMPA                          ;
191                           FAULT    LA,L,5      ;
192                           FAULT    HA,H,5      ;       % P5
193                           FAULT    LR,L,7      ;
194                           FAULT    HR,H,7      ;       % P7
195                 RETT                           ;
```



西蒙尼提供的这段代码展示了电路模拟器的片段，该模拟器通过比较观测到的行为与许多可能故障的模拟行为，可以自动定位Illiac-4并行计算机的硬件故障。西蒙尼说："这台大型计算机一共有64个处理器，全部用于针对各种可能发生的故障进行模拟。除个别情况外，从输出（见下方）来看，结果令人满意。注意插卡、芯片和特定故障类型已用记号代替。源代码里的FAULT宏汇集了多条机器指令，用来模拟4000多个并行模拟中一个模拟的故障，另外还会构建解读结果所需的数据表。糟糕的是，高级语言没有提供这方面的便利以实现这个层面的功能，这也是不可原谅的。"

```
#16 Card MSG001 dip F4 type 8-6594 output pin 4 stuck L
        Feeds Xmission line thru C39 terminated at PAT001 pin C33
```

1/24/86 Actually it's e : t1 ◊ f2 → t1 ◊ f2', succeeding if f2 fit e = f2' fit e

We only need it in apply, so we could put it in as a convenience feature like (but different from) coerce.

is this too strong? yes — we might need to do it again

This formulation also forces coerceF to evaluate successfully

This should be viewed as an alternative to keeping the :: rule for ◊

The :: rule also has a ⊗ clause. This is currently needed so that LET will work on formals defined with ⊗. This is not just a convenience, since the LET is supplied not by the programmer, but by the rule for λ or !. And coerce can't do it, because LET doesn't have a tough type.

Furthermore, we are going to add a clause

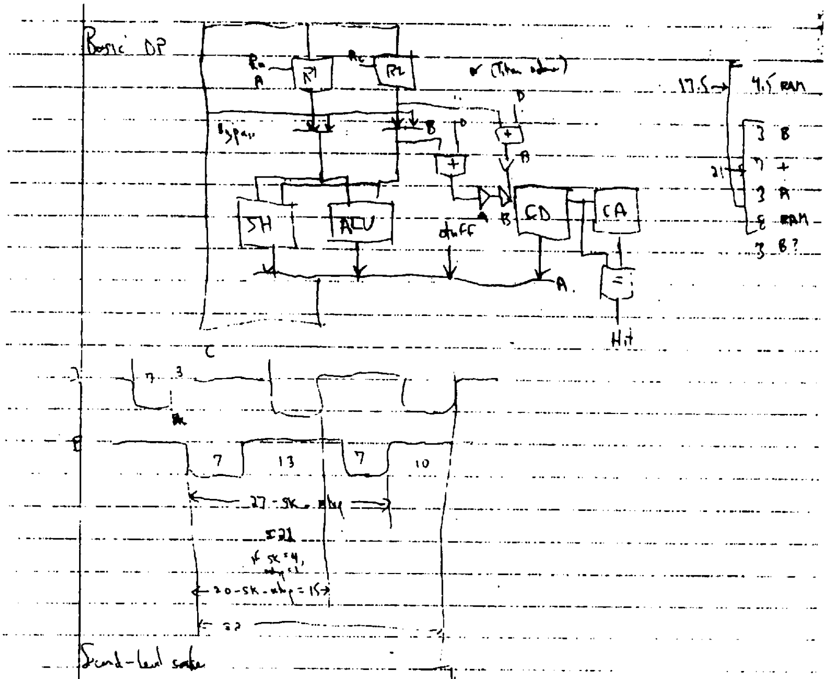
E :: typeOf t ⇒ E :: t

The currently proposed alternative is to ... function. The :: clause, however, would allow us to have keyword parameters to go to any function whose parameter type is a decl.

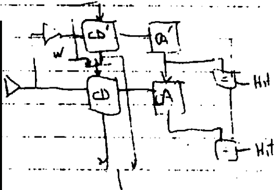
A related question is dropping of names in ⊗ and ⇒. The current rule is that ... doesn't drop names, but λ does for →, but not for ⇒. If we made a coercion from t1 ◊ f to t2 ◊ f if typeOf t1 = typeOf t2, we wouldn't need this coercion, this could be a :: rule. Or it could be if typeOf t1 = t2.



巴特勒·兰普森，数字设备公司系统研究中心高级工程师，他提供的这些笔记揭示了设计程序涉及的数学细节和思维过程。



Second-level cache



这张草图是巴特勒·兰普森提供的带二级缓存的高速CPU的初步设计，附带部分性能计算。其中流水线的时序要求参见第二页底部的计算。

1K x 32 cache
x 4.5 ns
takes only 8 1K x 4 chips
needs only 12 address bits, not 16

With 4KB, it about hit 90%. So effective time = .9×1 + .08×2 + .02×20
= .9 + .16 + .4 = 1.46



下面前3页是兰普森针对大规模分布式名称服务器的数据结构和不变式的初始笔记。后两页是修饰过的打印版本。

Conclusion – 20 is much too much. We need another level.

Try $.9\times1 + .095\times4 + .005\times20$:
$= .9 + .38 + .1 = 1.38$
Not much better

$.9\times1 + .08\times2 + .015\times5 + .005\times20$
$= .9 + .16 + .08 + .1 = 1.24$

Maybe the first scheme isn't so bad. Compare

$.8\times1 + .18\times2 + .02\times20$
$= .8 + .36 + .4 = 1.56$

Not much worse. The .8 is conservative from ?Cs: 90 measurements (4KB DM)

So ~ .5 cycles / cache ref will go into miss stalls.
Our VAX ?? does 2/3 read ?? /3 writes, roughly. So there are also per instruction.

B? secondary cache. 1 MByte = 64K × 32 × 4. So 4 banks of 64k. If a 4? clock is feasible, we can get a word every 10 ns from 128 chips. Shifting out the 4 words at 100 MHz would be a perfect ?? for a 128-bit word. Humble

The pipeline is

R ??	?	8-10 ns	
R ??	2		
R reg	?	4.5 R??, 3 bypass + latch	7.5/?
ALU ?/ cache	8	4 add/latch 4.5 R?n	12.5/16 ??
ALU / cache	?	4 + /latch	There's lots more time for data
W reg	8	this is where we can stop it easily	

On a stall, the B clock gets stopped, so the cache address (latched by B after adding) holds its stalled value. The instruction register holds at the next instruction. The write from the stalled instruction is stopped. This all seems fine. I'm not clear on what would happen if we tried to delay the write another cycle, ?? bypassing, in order to ?? more time for the stall.



If [illegible] are not too big, this must work. It may work inefficiently if the tree is unbalanced.

[illegible] — on FOP:.

14 Dec 84 Name server

```
TYPE
    D = UID × SU*                                                   directory
    S = UID → DC × [illegible]: N × up: (SN × parent: UID)*         server
    N = CHAR*                                                       name
    UN = N*                                                         universal name
    SN = UN                                                         server name
    DC = f: U → V × [illegible]: (N* | ⊥) × [illegible]: UID × [illegible]: TS   directory contents
    V = D | A | D × N | ...

PROC
    Contact: A → S
    FindServer: → S

Lookup: N × DC → V = dc(n)
Lookup*: N* × D → V = Lookup*(tail n*, Lookup(head n*, GetDC(d)))
GetDC: D → DC = IF x ≠ ⊥ THEN x ELSE GetDC(d.uid, tail d.s*) END
                WHERE x = head d.s*(d.uid)
GetDCHere: S × D → DC = IF x ≠ ⊥ THEN x ELSE GetDCHere(s, (d.uid, tail d.s*)) END
                WHERE x = IF s' = s THEN s'(d.uid) ELSE ⊥
                  WHERE s' = head d.s*
ULookup: UN × S → V = IF v ≠ ⊥ THEN Lookup*(n*, v)
                      ELSE ULookup(un, s') WHERE s' ∈ s.up
                      END
                        WHERE v, n* = ULHere(un, s)
ULHere: UN × S → V × N* =
        IF un = Λ THEN (⊥, Λ)
        ELSIF ∃ dc, uid: s(uid) = dc AND dc.h = un THEN ((uid, s), Λ)
        ELSE (v, (n*, n)) WHERE v, n* = ULHere(un', s) WHERE un = (un', n)
        END
```



INVARIANT

d = (uid, s*) ⇒ ∃ s ∈ s* : s(uid) ≠ ⊥ — Some listed server stores the directory

?∨ ∀s : S : s(uid) = ⊥ ?

∀s (∀s' ∈ s.up s'.bad < s.bad) ∨ s(root) ≠ ⊥ — Up defines a spanning tree of servers rooted in a set of servers that store the root

∀uid ∃s : s(uid) ≠ ⊥ ⇒ reachable(uid, root) — Any directory can be reached from the root

reachable : UID × DC → BOOL = ∃ n : dc(n).uid = uid ∨ reachable(uid, dc(n))

∀d : dc.h ≠ ⊥ ⇒ d.uid = ULookup(dc.h).uid — h is correct; this can't be weakened

WHERE dc = GetDC(d)

∃ (sn, uid) ∈ s.up : (x ≠ ⊥ AND Contact(x).bad < s.bad) OR s(root) ≠ ⊥

WHERE x = s(uid)(last sn)

?? ∀dc dc("uplinks") = (d, n) AND Lookup(n, d) = sn* AND

∀s ∃uid : s(uid) = sx.self ⇒ ∃ sn ∈ sn* :

Contact(ULookup(sn)) = s

Next : Change to h

&

Rotation

Changes to s* and

∀s₁, s₂, uid : s₁(uid) ≠ ⊥ AND s₂(uid) ≠ ⊥ ⇒ s₁(uid) ≈ s₂(uid)

All copies of D are the same, modulo distribution

17 Dec 84 Update distribution

TYPE U : (DC → DC) × UID . Pool = (SUN × U)*

Properties : ∀dc : u(dc) ≠ ⊥ — total

$u_1 \circ u_2 = u_2 \circ u_1$ — commute

?? $u \circ u = u$ — idempotent

PROC

CreateUpdate : DC × Uarg → U

StartUpdate : SX UID × Uarg → .

s.t. s(uid) = u(s₀(uid)) ∧ pool = pool₀ ∪ {u} pool₁ ∧ ad(uid) = ad₀(uid) ∪ {u}

AddSchedule (s(uid) ≠ ⊥ ⇒ ∃! (sn, u) ∈ pool₁ : sn ∈ S.self) ∧

s.self (sn, u') ∈ pool₁ ⇒ u' = u

WHERE u = CreateUpdate(s₀(uid), arg)

DoUpdate : · → .

s.t. ∃ (sun, u) ∈ pool₀ ∧ (sn, u) ∉ pool ∧ s(u.uid) = u(s₀(u.uid))

WHERE sun ∈ S₀.self



```
TYPE
  NAT = ****                            NATural number
  UID = NAT                             Universal IDentifier
  T = NAT                               Time; e.g., GMT (not site-dependent)
  TS = T X UID                          TimeStamp
  TX = T X extend: T                    Timeout eXpiration
  OK = yes | no

  DI = UID                              Directory Identifier
  SI = DI                               Server Identifier
  N = CHAR*                             Name
  DN = N X di: (DI | NIL)               Directory Name wrt d; invalid
                                         if di#NIL AND d(dn.n).di#dn.di
  FN = DI X DN*                         Full Name of a D from root di
  SN = FN X DN X TX                     Server Name; C.GetV(sn.fn, sn.dn.n):SA
  Link = FN X TX                        tx times out Link, as in DR
  LD = linked | direct                  kind of FN, or follow final link
  Lim = TX X LD                         MIN TX used in name lookup, used Link

  DR = DI                               D Reference; REP is DL.
(* A FN through dr is guaranteed valid only until dr^.br.tx *)
  BR = DI X N X TX | NIL                Back Reference; REP is TS->BL
  D = V                                 Directory; REP=REP V abstractly,
(* DC** concretely.  The other components of DC
   are really the values of names in d.v; e.g., d.di=d("&di"). *)
    X DI                                self
    X w: (DI->FN)                       well-known DIs (for old roots)
    X h: (FN X Lim)**                   help: FNs of d; may be invalid
(* protection and authentication data *)
    X RL
    X AA
    X K                                 Key for this dir viewed as a principal
(* time-stamps *)
    X lastTS: TS                        largest TS of any U to d
    X allUpTo: TS                       u IN REP d AND u.ts<d.allUpTo=>
                                        u IN FollowDR(d.di)
    X BR                                ds(d.br.di)(d.br.n).di=d.di

  V = L->(VV X TS)                      Value: tree with L arcs, TS nodes, Mk
  VV = Mk | V                            leaves. REP V=U** X TS, REP U**=W
  Mk = absent | present                 what to MaKe L* into in an update
  L = N | Tag | AV                      Label
  Tag = CHAR*                           labels fields of a d(n) value.
(* The value of the tag "&type" is the type of d(n). *)
  AV = BYTE*                            Atomic Value
  LTree = (L X LTree)**                 stripped V: tree, L arcs, empty leaves
  LT = L X TS
  LTL = LT* X L*                        designate V reached by the path
(* (Labels(lt*), l*), with TS matching lt*.ts. *)
  VD = FN X LTL X LD                    V Designator;
(* fn short for (fn, nullLTL, direct) and (fn, ltl) for (fn, ltl, direct). *)

  Vr = U** X TS                         REP V: set of updates (with REP W) X TS
  U = f: (V->V) X TS                    Update; REP=Y.
  W = Y**                               REP REP V: updatable with a Y
  Y = LT* X Mk                          REP U: set lt* component to mk
(* --------------- Types below this point are for DC/S/SU/E --------------- *)
  DL = DI X SN**                        Directory Locator, REP DR
  BL = DI X N X TX X Link               Back Locator. REP BR=TS->BL.
(* Normally br(ts)#% for 1 TS (0 for root), bl.link.fn^.di=bl.di. *)
  DC = D                                Directory Copy.
     X copies: SN**                     updated like a D component
(* A server s updates only sn's in copies with sn.dn.di=s.si. *)
     X inSN: BOOL                       updated like a D component
     X On
```



```
D: (* DIRECTORY *) MODULE
  EXPORTS [$napshot .. Resolve],
          FirstRoot, NewRoot, NewD, RemoveD, ChangeN,
          $FollowFN, $FollowDR
  IMPORTS R.GetRights, Env, V.(GetV, FrozenLink)
=
INVARIANT
1 ds(di)#% <=> BRPath(di) WHERE BRPath: DI->BOOL =
   LET dr=di,br=dr^.br,dr'=br.di; di=root OR dr'^(br.n).di=di AND BRPath(br.di))
        I.e., the D's in ds form a tree rooted in root whose arcs are DRs that
        are the reverse of the BR backpointers. D3 means that the tree you see
        can shift around if BRs are changing faster than allUpTo. REP is DC1.
2 dr'^(n):DR=dr => EXISTS tx: (dr'.di, n, tx)=dr^.br AND tx>=dr.tx
        I.e., each DR is pointed to by a BR with a longer timeout. REP is DC2.
3 d=dr^ => {u IN u** | u.ts<d.allUpTo} <= REP d <= u** WHERE u**=REP ds(dr.di)
        I.e., FollowDR gives an answer that includes all the updates to D
        before allUpTo, and any selection of those later. REP is SU1-2.
4 (fn, lim) IN d.h AND lim.tx>now => dr^ IS d AND dr.tx>now WHERE dr=fn^
        I.e., h entries are valid unless timed-out.
5 d.w(di)=fn => fn:FN AND fn.di=d.di AND fn^.di=di
        I.e., a d.w entry for a DI is a FN for it from d. Not enforced.
6 (fn.di=root OR FollowDR(root).w(fn.di)#%) AND
  ( (fn', n)<=fn => FollowDR(fn'^.di)(n):DR ) =>
  Get((fn, ltl, direct), {})#%
        I.e., Get looks up an FN to yield a D if its DI is root or is defined
        in root's w, and each N in its DN* can be looked up to yield a DR.

PROCEDURE
(* These implementations are not real. I.e., they describe what the procedures
   do, but aren't identical to any code in the system. These operations are
   actually implemented in NS and DC in the server, called remotely from the
   clerk. *)
```



兰普森提供了一种数据类型为头等（first-class）数值的编程语言的语法和语义示例。其中语义的一部分在第二页上以该语言简便形式（“糖”）的方式给出，另一部分由最后一页的逻辑推理规则给出。

Pebble summary 26 January 1966

Syntax

Binary operators: ', (2) ~ (3)–> (4) * (5)

Everything associates to the right

E = N
| N : T (6)
| £ T IN E (1)
| LET B IN E (1)
| IMPORT B IN E (1)
| F ° E (7)

| N:~E | '(N_1, ...'):~E | N'(T'):~E
| REC D_1~E_1, ... | REC N_1'(T_1:~E_1'), ...
| B_1 ; B_2 (2) ??
| B$N (6)
| E WHERE B (1)
| D–»T | D ** E (4, 5)
| E_1. N | E_1 N'(E_2') (6)
| E: N: (6)

| '(')
| { E } | T{ E }
| *prefixOp* E | E_1 *infixOp* E_2 | E *postfixOp*
| E_1 AND E_2 | E_1 OR E_2 (5, 4)
| IF (E_0 => E_1) '| ... '|=> E_2 | IF (E_0 => E_1) '| ... FI
| CASE E_0 OF (E_1 '(D ') => E_2) '| ... |=> E_3 END
| E_0 BUT (E_1 '(D ') => E_2) '| ... END
| LOOP E END
| FOR N IN E_1 DO E_2 THEN E_3 END
| FOR N IN E_1 WITH E_4 COLLECT E_2 THEN E_3 END



Pebble summary 26 January 1986

Sugar

<table>
<tr><th>Write</th><th>For</th><th>Provided</th></tr>
<tr><td>N:~E</td><td>N: t~E</td><td>E:> t</td></tr>
<tr><td>'(N_1, N_2, ...'):~E</td><td>N 1:~fst E, '(N_2, ...'):~snd E</td><td></td></tr>
<tr><td>N'(?T'):~E</td><td>N:~£ ?T IN E</td><td></td></tr>
<tr><td>£ D IN E</td><td>£ D—>t IN E</td><td>LET newc*d IN E:>t
and newc not in t</td></tr>
<tr><td>£ IN E</td><td>£ void IN E</td><td></td></tr>
<tr><td>REC D_1~E_1, ...</td><td>'(fix '(D_1 * ...') '(£ B′: '(D_1 * ...')
IN LET B′ IN '(E_1, ...')')</td><td></td></tr>
<tr><td>REC N_1'(T_1'):~E_1, ...</td><td>REC N_1: t_1~*£ T_1 IN E_1, ...</td><td></td></tr>
<tr><td>B_1; B_2</td><td>B_1, LET B_1 IN B_2</td><td>??</td></tr>
<tr><td>B$N</td><td>IMPORT B IN N</td><td></td></tr>
<tr><td>E WHERE B</td><td>LET B IN E</td><td></td></tr>
<tr><td>D—»T</td><td>D%'(£ D—>type IN T')</td><td></td></tr>
<tr><td>D ** T</td><td>D◇'(£ D—>type IN T')</td><td></td></tr>
<tr><td>E_1. N</td><td>'(snd xt^{-1} t')^$N'($E_1$')</td><td>$E_1$:> t</td></tr>
<tr><td>E_1. N'(E_2')</td><td>'(snd xt^{-1} t')^$N'($E_1$, E_2')</td><td>E_1:> t</td></tr>
<tr><td>E: T:</td><td>T: type ** E: T@'(£ T: type IN T')</td><td></td></tr>
<tr><td>()</td><td>nil</td><td></td></tr>
<tr><td>{ E }
or</td><td>'(MkSet t') E</td><td>fst E:> t</td></tr>
<tr><td>T{ E, ...}</td><td>SingleSet'(T, E') u ...</td><td></td></tr>
<tr><td>T{}</td><td>EmptySet T</td><td></td></tr>
<tr><td>{ E, ...}</td><td>SingleSet'(t, E') u ...</td><td>E:> t</td></tr>
<tr><td>prefixOp E</td><td>E . op'("prefixOp"')</td><td></td></tr>
<tr><td>E_1 infixOp E_2</td><td>E_1 . op'("infixOp"') '(E_2')</td><td></td></tr>
<tr><td>E postfixOp</td><td>E . op'("postfixOp"')</td><td></td></tr>
<tr><td>E_1 AND E_2</td><td>IF E_1 => E_2 |=> false</td><td></td></tr>
<tr><td>E_1 OR E_2</td><td>IF E_1 => true |=> E_2</td><td></td></tr>
</table>

Pebble 86 Summary

Rules

Introduction £ :

{ t== d_1 —> t_2 or t== d_1 % f_2, f_2! newc(n)~> t_2 },
rho(depth)+1= n, rho[depth=n] |- LET newc(n)$*_{d1}$ IN t_2~E :: t_2 => e

(£ T IN E) :> t => cl([], e, n)

T :: type

N: T :: type => N:t

Elimination N apply LET IMPORT

rho(N)== t~ e' , e' ~» e

N :> t => e

{ F :: t_1 —> t_2 or F :: t_1 % f_2, $f_2$$*_{t1\text{->}type}$ E_1 => t_2 },
{ E_1 :: t_1=> e_1 or **E_1 :> t_1', coerceF! (t_1', t_1)~> f_1, $f_1$$*_{t1'\text{->}t1}$ E_1 => e_1** },
{ f! e_1 ~> e_2 **else** f! e_1= e_2}

F ° E_1 :> t_2=> e_2

B :: void, E :: t=> e
or B :: (N: t_0)=> b, rho[N=t_0~b] |- E :: t=> e
or B :: d_1 ◇ f_2, snd B :> d_2 => b_2, LET fst B IN LET $b_2$$*_{d2}$ IN E :: t=> e

LET B IN E :> t => e

B :> d=> b, rho_0 |- LET b$*_d$ IN E :: t=> e

IMPORT B IN E :> t

Auxiliary :: * ~> ~»

E :> t
or E :: t @ f_2
or E :: t', t u t'=t could be coercion
or t== N: t', E :: t' could be coercion
or t== t_1 ◇ f_2, fst E :: t_1, snd E :: f_2 fst E
or t== t_1 % f_2, E :: t_1 % f_2', t_1== t_1' / s_1, for all e_1 IN s_1: (f_2! e_1~> t_2, f_2'! e_1~> t_2', t_2=t_2')

E :: t





CP/M操作系统的创始人加里·基尔代尔为他最近的项目——光盘百科全书检索系统的示意图作了如下说明："此图显示了基于字符的知识检索系统菜单面板之间的相互关系。每个屏幕左边的方块显示最初（和变更后）的功能键绑定。箭头显示了从一个面板到另一个面板的转变。实施环节新增了菜单，注释显示的是绘制面板的C语言子程序名称。下面两页图表形象地展示了产品开发中的KRS结构。"

> 下面5页是比尔·盖茨和保罗·艾伦在1975年写的部分8080 BASIC原始代码。这个历史性的程序是专为微型计算机（MITS公司开发的Altair）打造的第一个高级语言。这是程序开头的注释，主要描述内存是怎么配置的。

```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)-1 21:12 21-FEB-78 PAGE 1-4
F3        MAC      17-FEB-78 19:18            SOME EXPLANATION

        SUBTTL  SOME EXPLANATION
        COMMENT *
        ALTAIR BASIC CONFIGURES MEMORY AS FOLLOWS:
        LOW LOCATIONS     (NON-TIMESHARING VERSION)
                RST       SUBROUTINES
                          0       STARTUP
                                  INITIALLY A JMP TO THE INITIALIZATION CODE
                                  BUT CHANGED TO A JMP TO READY.
                                  RESTARTING THE MACHINE AT 0 DURING PROGRAM
                                  EXECUTION CAN LEAVE THINGS MESSED UP.

                          1       SYNCHK
                                  A CHECK IS MADE TO MAKE SURE THE
                                  CHARACTER POINTER POINTS AT A SPECIFIC
                                  CHARACTER. IF NOT THE "SYNTAX ERROR"
                                  ROUTINE IS CALLED. IF SO,
                                  THE CHRGET RST IS DROPPED INTO SO
                                  THE CHARACTER AFTER THE MATCHED
                                  ONE WILL BE PUT IN [A] AND
                                  THE CONDITION CODES WILL REFLECT THIS
                                  EXAMPLE: SYNCHK THENTK (THE MATCH CHARACTER IS
                                  GIVEN IN THE LOCATION AFTER THE RST)
                                  WOULD CHECK TO MAKE SURE [H,L] POINTED TO A THENTK
                                  AND IF SO FETCH THE NEXT CHARACTER INTO [A],
                                  IF NOT, A "SYNTAX ERROR" WOULD BE GIVEN.

                          2       CHRGET
                                  USING [H,L] AS THE TEXT POINTER
                                  THE TEXT POINTER IS INCREMENTED
                                  AND THE NEXT CHARACTER IS FETCHED INTO [A]
                                  IF THE CHARACTER IS A " " IT IS SKIPPED
                                  OVER AND THE NEXT CHARACTER IS FETCHED.
                                  THE STATEMENT TERMINATORS ":" AND 0
                                  LEAVE THE ZERO FLAG SET.
                                  THE NUMERICS "0" THROUGH "9" LEAVE THE CARRY
                                  FLAG SET. THE CURRENT CHARACTER CAN BE
                                  REFETCHED INTO [A] BY DOING A MOV A,M.
                                  IF THE CONDITION CODES MUST BE SET UP AGAIN
                                  DCX H,CHRGET WILL WORK. IT IS VERY DIFFICULT
                                  TO REEXAMINE THE CHARACTER BEFORE THE CURRENT
                                  ONE SINCE SPACES MAY BE IN-BETWEEN.
                                  DCX H,DCX H,CHRGET WILL NOT ALWAYS WORK.

                          3       OUTCHR
                                  THE CHARACTER IN [A] IS PRINTED ON
                                  THE USER'S TERMINAL. [A] AND THE
                                  CONDITION CODES ARE PRESERVED.

                          4       COMPAR
                                  [D,E] AND [H,L] ARE COMPARED AS UNSIGNED

BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)-1 21:12 21-FEB-78 PAGE 1-5
F3        MAC      17-FEB-78 19:18            SOME EXPLANATION

                                  DOUBLE-BYTE INTEGERS. CARRY IS SET IF
                                  [H,L] IS LESS THAN [D,E]. ZERO IS SET IF THEY
                                  ARE EQUAL. [A] IS SMASHED. THE ONLY DEFINITE
                                  THING THAT CAN BE SAID ABOUT [A] ON RETURN
                                  IS THAT IF THE ZERO FLAG IS SET, [A] WILL
                                  EQUAL 0.

                          5       FSIGN
                                  THE FAC (FLOATING ACCUMULATOR)
                                  WHICH IS USED TO STORE NUMERIC RESULTS
                                  IS CHECKED TO SEE WHAT SIGN ITS
                                  VALUE HAS.

                          6       PUSHM
                                  A DOUBLE BYTE QUANTITY POINTED
                                  TO BY [H,L] IS PUSHED ONTO THE
                                  STACK. [B,C] IS SET EQUAL TO THE
                                  VALUE PUSHED. [H,L] IS INCREMENTED BY TWO.

                          7       IN THE 4K VERSION RST 7 IS UNUSED AND THE LOCATIONS
                                  ASSOCIATED WITH IT ARE USED TO CONTINUE
                                  THE CODE FOR RST 6. IN THE 8K A JMP IS MADE
                                  AROUND THE FIRST THREE RST 7 LOCATIONS
                                  DURING RST 6 EXECUTION. RST 7 INITIALLY
                                  CONTAINS A RET, BUT THE USER CAN CHANGE IT TO
                                  A JMP TO AN INTERRUPT SERVICE ROUTINE.

        (IN THE TIMESHARING VERSION, THE MONITOR GOES
         INTO THE LOW AREA, AND THE BASIC INTERPRETER
         FOLLOWS IT. THE RESTART ROUTINES DOCUMENTED
         ABOVE ARE CALLS TO SUBROUTINES INSTEAD.)

        FUNCTION DISPATCH ADDRESSES
                FUNDSP CONTAINS THE ADDRESSES OF THE
                FUNCTION ROUTINES IN THE ORDER OF THE
                FUNCTION NAMES IN THE CRUNCH LIST.
                THE FUNCTIONS THAT TAKE MORE THAN ONE ARGUMENT
                ARE AT THE END. SEE THE EXPLANATION AT ISFUN.

        THE OPERATOR TABLE
                THE OPTAB TABLE CONTAINS AN OPERATORS PRECEDENCE
                FOLLOWED BY THE ADDRESS OF THE ROUTINE TO PERFORM
                THE OPERATION. THE INDEX INTO THE
                OPERATOR TABLE IS MADE BY SUBTRACTING OFF THE CRUNCH VALUE
                OF THE LOWEST NUMBERED OPERATOR. THE ORDER
                OF OPERATORS IN THE CRUNCH LIST AND IN OPTAB IS IDENTICAL.
                THE PRECEDENCES ARE ARBITRARY EXCEPT FOR THEIR
                COMPARATIVE SIZES. NOTE THAT THE PRECEDENCE FOR
                UNARY OPERATIONS SUCH AS NOT AND NEGATION ARE
                SETUP SPECIALLY WITHOUT USING A TABLE.

        THE RESERVED WORD OR CRUNCH LIST
                WHEN A COMMAND OR PROGRAM LINE IS TYPED IN
                IT IS STORED IN BUF. AS SOON AS THE WHOLE LINE
                HAS BEEN TYPED IN (INLIN RETURNS) CRUNCH IS
```






```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)-1 21:12 21-FEB-78 PAGE 1-6
F3      MAC      17-FEB-78 19:18               SOME EXPLANATION

473                    03480                  CALLED TO CONVERT ALL RESERVED WORDS TO THEIR
474                    03490                  CRUNCH VALUES. THIS REDUCES THE SIZE OF THE
475                    03500                  PROGRAM AND SPEEDS UP EXECUTION BY ALLOWING
476                    03510                  TABLE DISPATCHES TO PERFORM FUNCTIONS,STATEMENTS,
477                    03520                  AND OPERATIONS. THIS IS BECAUSE ALL THE STATEMENT
478                    03530                  NAMES ARE STORED CONSECUTIVELY IN THE CRUNCH LIST.
479                    03540                  WHEN A MATCH IS FOUND BETWEEN A STRING
480                    03550                  OF CHARACTERS AND A WORD IN THE CRUNCH LIST
481                    03560                  THE ENTIRE TEXT OF THE MATCHED WORD IS TAKEN OUT OF
482                    03570                  THE INPUT LINE AND A RESERVED WORD TOKEN IS PUT
483                    03580                  IN ITS PLACE. A RESERVED WORD TOKEN IS ALWAYS EQUAL
484                    03590                  TO OCTAL 200 PLUS THE POSITION OF THE MATCHED WORD
485                    03600                  IN THE CRUNCH LIST.
486                    03610
487                    03620          STATEMENT DISPATCH ADDRESSES
488                    03630                  WHEN A STATEMENT IS TO BE EXECUTED, THE FIRST
489                    03640                  CHARACTER OF THE STATEMENT IS EXAMINED
490                    03650                  TO SEE IF IT IS LESS THAN THE RESERVED
491                    03660                  WORD TOKEN FOR THE LOWEST NUMBERED STATEMENT NAME.
492                    03670                  IF SO, THE "LET" CODE IS CALLED TO
493                    03680                  TREAT THE STATEMENT AS AN ASSIGNMENT STATEMENT.
494                    03690                  OTHERWISE A CHECK IS MADE TO MAKE SURE THE
495                    03700                  RESERVED WORD NUMBER IS NOT TOO LARGE TO BE A
496                    03710                  STATEMENT TYPE NUMBER. IF NOT THE ADDRESS
497                    03720                  TO DISPATCH TO IS FETCHED FROM STMDSP (THE STATEMENT
498                    03730                  DISPATCH TABLE) USING THE RESERVED WORD
499                    03740                  NUMBER FOR THE STATEMENT TO CALCULATE AN INDEX INTO
500                    03750                  THE TABLE.
501                    03760
502                    03770          ERROR MESSAGES
503                    03780                  WHEN AN ERROR CONDITION IS DETECTED
504                    03790                  [E] MUST BE SET UP TO INDICATE WHICH ERROR
505                    03800                  MESSAGE IS APPROPRIATE AND A BRANCH MUST BE MADE
506                    03810                  TO ERROR. THE STACK WILL BE RESET AND ALL
507                    03820                  PROGRAM CONTEXT WILL BE LOST. VARIABLES
508                    03830                  VALUES AND THE ACTUAL PROGRAM REMAIN INTACT.
509                    03840                  ONLY THE VALUE OF [E] IS IMPORTANT WHEN
510                    03850                  THE BRANCH IS MADE TO ERROR. [E] IS USED AS AN
511                    03860                  INDEX INTO ERRTAB WHICH GIVES THE TWO
512                    03870                  CHARACTER ERROR MESSAGE THAT WILL BE PRINTED ON THE
513                    03880                  USER'S TERMINAL.
514                    03890
515                    03900          IMPURE STORAGE
516                    03910                  ALL TEMPORARIES,FLAGS,POINTERS, THE BUFFER AREA,
517                    03920                  THE FLOATING ACCUMULATOR,AND ANYTHING ELSE THAT
518                    03930                  IS USED TO STORE A CHANGING VALUE SHOULD BE LOCATED
519                    03940                  IN THIS AREA. CARE MUST BE MADE IN MOVING LOCATIONS
520                    03950                  IN THIS AREA SINCE THE JUXTAPOSITION OF TWO LOCATIONS
521                    03960                  IS OFTEN DEPENDED UPON.
522                    03970
523                    03980          TEXTUAL MESSAGES
524                    03990                  CONSTANT MESSAGES ARE STORED HERE. UNLESS
525                    04000                  THE CODE TO CHECK IF A STRING MUST BE COPIED
526                    04010                  IS CHANGED THESE STRINGS MUST BE STORED ABOVE
527                    04020                  DSCTMP, OR ELSE THEY WILL BE COPIED BEFORE
```

```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)-1 21:12 21-FEB-78 PAGE 1-7
F3      MAC      17-FEB-78 19:18               SOME EXPLANATION

528                    04030                  THEY ARE PRINTED.
529                    04040
530                    04050          FNDFOR
531                    04060                  MOST SMALL ROUTINES ARE FAIRLY SIMPLE
532                    04070                  AND ARE DOCUMENTED IN PLACE. FNDFOR IS
533                    04080                  USED FOR FINDING "FOR" ENTRIES ON
534                    04090                  THE STACK. WHENEVER A "FOR" IS EXECUTED AN
535                    04100                  16 BYTE ENTRY IS PUSHED ONTO THE STACK.
536                    04110                  BEFORE THIS IS DONE, HOWEVER, A CHECK
537                    04120                  MUST BE MADE TO SEE IF THERE
538                    04130                  ARE ANY "FOR" ENTRIES ALREADY ON THE STACK
539                    04140                  FOR THE SAME LOOP VARIABLE. IF SO, THAT "FOR" ENTRY
540                    04150                  AND ALL OTHER "FOR" ENTRIES THAT WERE MADE AFTER IT
541                    04160                  ARE ELIMINATED FROM THE STACK. THIS IS SO A
542                    04170                  PROGRAM THAT JUMPS OUT OF THE MIDDLE
543                    04180                  OF A "FOR" LOOP AND THEN RESTARTS THE LOOP AGAIN
544                    04190                  AND AGAIN WON'T USE UP 18 BYTES OF STACK
545                    04200                  SPACE EVERY TIME. THE "NEXT" CODE ALSO
546                    04210                  CALLS FNDFOR TO SEARCH FOR A "FOR" ENTRY WITH
547                    04220                  THE LOOP VARIABLE IN
548                    04230                  THE "NEXT". AT WHATEVER POINT A MATCH IS FOUND
549                    04240                  THE STACK IS RESET. IF NO MATCH IS FOUND A
550                    04250                  "NEXT WITHOUT FOR" ERROR OCCURS. GOSUB EXECUTION
551                    04260                  ALSO PUTS A 5 BYTE ENTRY ON STACK.
552                    04270                  WHEN A RETURN IS EXECUTED FNDFOR IS
553                    04280                  CALLED WITH A VARIABLE POINTER THAT CAN'T
554                    04290                  BE MATCHED. WHEN "FNDFOR" HAS RUN
555                    04300                  THROUGH ALL THE "FOR" ENTRIES ON THE STACK
556                    04310                  IT RETURNS AND THE RETURN CODE MAKES
557                    04320                  SURE THE ENTRY THAT WAS STOPPED
558                    04330                  ON IS A GOSUB ENTRY. THIS ASSURES THAT
559                    04340                  IF YOU GOSUB TO A SECTION OF CODE
560                    04350                  IN WHICH A FOR LOOP IS ENTERED BUT NEVER
561                    04360                  EXITED THE RETURN WILL STILL BE
562                    04370                  ABLE TO FIND THE MOST RECENT
563                    04380                  GOSUB ENTRY. THE "RETURN" CODE ELIMINATES THE
564                    04390                  "GOSUB" ENTRY AND ALL "FOR" ENTRIES MADE AFTER
565                    04400                  THE GOSUB ENTRY.
566                    04410          NON-RUNTIME STUFF
567                    04420                  THE CODE TO INPUT A LINE,CRUNCH IT,GIVE ERRORS,
568                    04430                  FIND A SPECIFIC LINE IN THE PROGRAM,
569                    04440                  PERFORM A "NEW","CLEAR", AND "LIST" ARE
570                    04450                  ALL IN THIS AREA. GIVEN THE EXPLANATION OF
571                    04460                  PROGRAM STORAGE GIVEN BELOW THESE ARE
572                    04470                  ALL STRAIGHTFORWARD.
573                    04480
574                    04490          NEWSTT
575                    04500
576                    04510                  WHENEVER A STATEMENT FINISHES EXECUTION IT
577                    04520                  DOES A "RET" WHICH TAKES
578                    04530                  EXECUTION BACK TO NEWSTT. STATEMENTS THAT
579                    04540                  CREATE OR LOOK AT SEMI-PERMANENT STACK ENTRIES
580                    04550                  MUST GET RID OF THE RETURN ADDRESS OF NEWSTT AND
581                    04560                  JMP TO NEWSTT WHEN DONE. NEWSTT ALWAYS
582                    04570                  CHRGETS THE FIRST CHARACTER AFTER THE STATEMENT
```

```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF     MACRO 550(272)-1 21:12 21-FEB-78 PAGE 1-8
F3      MAC     17-FEB-78 19:18          SOME EXPLANATION

   583          04580                      MADE BEFORE DISPATCHING. WHEN RETURNING
   584          04590                      BACK TO NEWSTT THE ONLY THING THAT
   585          04600                      MUST BE SET UP IS THE TEXT POINTER IN
   586          04610                      [H,L]. NEWSTT WILL CHECK TO MAKE SURE
   587          04620                      [H,L] IS POINTING TO A STATEMENT TERMINATOR.
   588          04630                      IF A STATEMENT SHOULDN'T BE PERFORMED UNLESS
   589          04640                      IT IS PROPERLY FORMATTED (I.E. "NEW") IT CAN
   590          04650                      SIMPLY DO A "RNZ" AFTER READING ALL OF
   591          04660                      ITS ARGUMENTS. SINCE THE ZERO FLAG
   592          04670                      BEING OFF INDICATES THERE IS NOT
   593          04680                      A STATEMENT TERMINATOR NEWSTT WILL
   594          04690                      DO THE JMP TO THE "SYNTAX ERROR"
   595          04700                      ROUTINE. IF A STATEMENT SHOULD BE STARTED
   596          04710                      OVER IT CAN DO LHLD TEMP,RET SINCE THE [H,L]
   597          04720                      AT NEWSTT IS ALWAYS STORED IN TEMP. OF COURSE
   598          04730                      CARE MUST BE TAKEN THAT NO ROUTINE
   599          04740                      THAT SMASHES TEMP HAS BEEN CALLED.
   600          04750                      THE ^C CODE STORES TEMP IN OLDTXT AND CURLIN (THE
   601          04760                      CURRENT LINE NUMBER) IN OLDLIN SINCE THE ^C CHECK
   602          04770                      IS MADE BEFORE THE STATEMENT POINTED TO IS
   603          04780                      EXECUTED. "STOP" AND "END" STORE THE TEXT POINTER
   604          04790                      IN [H,L] WHICH POINTS AT THEIR TERMINATING
   605          04800                      CHARACTER IN OLDTXT.
   606          04810
   607          04820               STATEMENT CODE
   608          04830                      THE INDIVIDUAL STATEMENT CODE COMES
   609          04840                      NEXT. THE APPROACH USED IN EXECUTING EACH
   610          04850                      STATEMENT IS DOCUMENTED IN THE STATEMENT CODE
   611          04860                      ITSELF.
   612          04870
   613          04880               FRMEVL, THE FORMULA EVALUATOR
   614          04890                      GIVEN AN [H,L] POINTING TO THE STARTING
   615          04900                      CHARACTER OF A FORMULA FRMEVL
   616          04910                      EVALUATES THE FORMULA AND LEAVES
   617          04920                      THE VALUE IN THE FLOATING ACCUMULATOR (FAC).
   618          04930                      [H,L] IS RETURNED POINTING TO THE FIRST CHARACTER
   619          04940                      THAT COULD NOT BE INTERPRETED AS PART OF THE
   620          04950                      FORMULA. THE ALGORITHM USES THE STACK
   621          04960                      TO STORE TEMPORARY RESULTS:
   622          04970
   623          04980                              0, PUT A DUMMY PRECEDENCE OF ZERO ON
   624          04990                                      THE STACK.
   625          05000                              1, READ LEXEME (CONSTANT,FUNCTION,
   626          05010                                      VARIABLE,FORMULA IN PARENS)
   627          05020                                      AND TAKE THE LAST PRECEDENCE VALUE
   628          05030                                      OFF THE STACK.
   629          05040                              2, SEE IF THE NEXT CHARACTER IS AN OPERATOR
   630          05050                                      IF NOT, RETURN. THIS MAY CAUSE
   631          05060                                      OPERATOR APPLICATION OR AN ACTUAL
   632          05070                                      RETURN FROM FRMEVL.
   633          05080                              3, IF IT IS, SEE WHAT PRECEDENCE IT HAS
   634          05090                                      AND COMPARE IT TO THE PRECEDENCE
   635          05100                                      OF THE LAST OPERATOR ON THE STACK
   636          05110                              4, IF = OR LESS REMEMBER THE TEXT
   637          05120                                      POINTER AT THE START OF THIS OPERATOR

BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF     MACRO 550(272)-1 21:12 21-FEB-78 PAGE 1-9
F3      MAC     17-FEB-78 19:18          SOME EXPLANATION

   638          05130                                      AND DO A RETURN TO CAUSE
   639          05140                                      APPLICATION OF THE LAST OPERATOR.
   640          05150                                      EVENTUALLY RETURN TO STEP 2
   641          05160                                      BY RETURNING TO RETAOP.
   642          05170                              5, IF GREATER PUT THE LAST PRECEDENCE
   643          05180                                      BACK ON, SAVE THE CURRENT
   644          05190                                      TEMPORARY RESULT, OPERATOR ADDRESS
   645          05200                                      AND PRECEDENCE AND RETURN TO STEP 1.
   646          05210
   647          05220                      RELATIONAL OPERATORS ARE ALL HANDLED THROUGH
   648          05230                      A COMMON ROUTINE. SPECIAL
   649          05240                      CARE IS TAKEN TO DETECT TYPE MISMATCHES SUCH AS 3+"F"
   650          05250
   651          05260               EVAL -- THE ROUTINE TO READ A LEXEME
   652          05270                      EVAL CHECKS FOR THE DIFFERENT TYPES OF
   653          05280                      ENTITIES IT IS SUPPOSED TO DETECT.
   654          05290                      LEADING PLUSES ARE IGNORED.
   655          05300                      DIGITS AND "." CAUSE FIN (FLOATING INPUT)
   656          05310                      TO BE CALLED. FUNCTION NAMES CAUSE THE
   657          05320                      FORMULA INSIDE THE PARENTHESES TO BE EVALUATED
   658          05330                      AND THE FUNCTION ROUTINE TO BE CALLED. VARIABLE
   659          05340                      NAMES CAUSE PTRGET TO BE CALLED TO GET A POINTER
   660          05350                      TO THE VALUE, AND THEN THE VALUE IS PUT INTO
   661          05360                      THE FAC. AN OPEN PARENTHESIS CAUSES FRMEVL
   662          05370                      TO BE CALLED (RECURSIVELY), AND THE ")" TO
   663          05380                      BE CHECKED FOR. UNARY OPERATORS (NOT AND
   664          05390                      NEGATION) PUT THEIR PRECEDENCE ON THE STACK
   665          05400                      AND ENTER FORMULA EVALUATION AT STEP 1, SO
   666          05410                      THAT EVERYTHING UP TO AN OPERATOR GREATER THAN
   667          05420                      THEIR PRECEDENCE OR THE END OF THE FORMULA
   668          05430                      WILL BE EVALUATED. WHEN FRMEVL DOES A RETURN
   669          05440                      BECAUSE IT SEES AN OPERATOR OF HIGHER PRECEDENCE
   670          05450                      IT DOES NOT PASS THE TEXT POINTER IN [H,L], SO
   671          05460                      AFTER THE UNARY OPERATION HAS BEEN PERFORMED
   672          05470                      ON THE FAC THE TEXT POINTER MUST BE FETCHED FROM
   673          05480                      A TEMPORARY LOCATION THAT FRMEVL USES AND
   674          05490                      A RETURN BACK TO FRMEVL DONE.
   675          05500
   676          05510               DIMENSION AND VARIABLE SEARCHING
   677          05520                      SPACE IS ALLOCATED FOR VARIABLES AS THEY ARE
   678          05530                      ENCOUNTERED. THUS "DIM" STATEMENTS MUST BE
   679          05540                      EXECUTED TO HAVE EFFECT. 6 BYTES ARE ALLOCATED
   680          05550                      FOR EACH SIMPLE VARIABLE, WHETHER IT IS A STRING,
   681          05560                      NUMBER OR USER DEFINED FUNCTION. THE FIRST TWO
   682          05570                      BYTES GIVE THE NAME OF THE VARIABLE AND THE LAST FOUR
   683          05580                      GIVE ITS VALUE. [VARTAB] GIVES THE FIRST LOCATION
   684          05590                      WHERE A SIMPLE VARIABLE NAME IS FOUND AND [ARYTAB]
   685          05600                      GIVES THE LOCATION TO STOP SEARCHING FOR SIMPLE
   686          05610                      VARIABLES. A "FOR" ENTRY HAS A TEXT POINTER
   687          05620                      AND A POINTER TO A VARIABLE VALUE SO NEITHER
   688          05630                      THE PROGRAM OR THE SIMPLE VARIABLES CAN BE
   689          05640                      MOVED WHILE THERE ARE ACTIVE "FOR" ENTRIES ON THE STACK.
   690          05650                      USER DEFINED FUNCTION VALUES ALSO CONTAIN
   691          05660                      POINTERS INTO SIMPLE VARIABLE SPACE SO NO USER-DEFINED
   692          05670                      FUNCTION VALUES CAN BE RETAINED IF SIMPLE VARIABLES
```



```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)=1 21:12 21-FEB-78 PAGE 1-10
F3      MAC      17-FEB-78 19:18      SOME EXPLANATION

                ARE MOVED. ADDING A SIMPLE VARIABLE
                ADDING SIX TO ARYTAB AND STREND, BLOCK TRANSFERING
                THE ARRAY VARIABLES UP BY SIX AND MAKING SURE THE
                NEW [STREND] IS NOT TOO CLOSE TO THE STACK.
                THIS MOVEMENT OF ARRAY VARIABLES MEANS
                THAT NO POINTER TO AN ARRAY WILL STAY VALID WHEN
                NEW SIMPLE VARIABLES CAN BE ENCOUNTERED. THIS IS
                WHY ARRAY VARIABLES ARE NOT ALLOWED "FOR"
                LOOP VARIABLES. SETTING UP A NEW ARRAY VARIABLE
                MERELY INVOLVES BUILDING THE DESCRIPTOR,
                UPDATING STREND, AND MAKING SURE THERE IS
                STILL ENOUGH ROOM BETWEEN STREND AND THE
                STACK. WITHOUT MULTIPLE DIMENSIONS THE FORMAT
                OF AN ARRAY VARIABLE IS SIMPLY:
                        SECOND CHARACTER
                        FIRST CHARACTER
                        NUMBER OF BYTES USED BY VALUES
                        VALUES
                THE FORMAT WHEN MULTIPLY DIMENSIONED VARIABLES
                ARE ALLOWED IS DESCRIBED IN THE "MULDIM" CODE.
                PTRGET, THE ROUTINE WHICH RETURNS A POINTER
                TO A VARIABLE VALUE, HAS TWO IMPORTANT FLAGS. ONE IS
                "DIMFLG" WHICH INDICATES WHETHER "DIM" CALLED PTRGET
                OR NOT. IF SO, NO PRIOR ENTRY FOR THE VARIABLE IN
                QUESTION SHOULD BE FOUND, AND THE INDEX INDICATES
                HOW MUCH SPACE TO SET ASIDE. SIMPLE VARIABLES CAN
                BE "DIMENSIONED", BUT THE ONLY EFFECT WILL BE TO
                SET ASIDE SPACE FOR THE VARIABLE IF IT HASN'T BEEN
                ENCOUNTERED YET. THE OTHER IMPORTANT FLAG IS SUBFLG
                WHICH INDICATES WHETHER A SUBSCRIPTED VARIABLE SHOULD BE
                ALLOWED IN THE CURRENT CONTEXT. IF SUBFLG IS NON-ZERO
                THE OPEN PARENTHESIS FOR A SUBSCRIPTED VARIABLE
                WILL NOT BE SCANNED BY PTRGET, AND PTRGET WILL RETURN
                WITH A TEXT POINTER POINTING TO THE "("; IF
                THERE WAS ONE.
        STRINGS
                IN THE VARIABLE TABLE STRINGS ARE STORED JUST LIKE
                NUMERIC VARIABLES. SIMPLE STRINGS HAVE FOUR VALUE
                BYTES WHICH ARE INITIALIZED TO ALL ZEROS (WHICH
                REPRESENTS THE NULL STRING). THE ONLY DIFFERENCE
                IN HANDLING IS THAT WHEN PTRGET SEES A "$" AFTER THE
                NAME OF A VARIABLE, PTRGET SETS VALTYP TO ONE AND TURNS
                ON THE MSB (MOST-SIGNIFICANT-BIT) OF THE VALUE OF
                THE FIRST CHARACTER OF THE VARIABLE NAME.
                HAVING THIS BIT ON IN THE NAME OF THE VARIABLE ENSURES
                THAT THE SEARCH ROUTINE WILL NOT MATCH
                'A' WITH 'A$' OR 'A$' WITH 'A'. THE MEANING OF
                THE FOUR VALUE BYTES ARE:
                        LOW
                                LENGTH OF THE STRING
                                UNUSED
                                LOW 8 BITS
                                HIGH 8 BITS  OF THE ADDRESS
                                        OF THE CHARACTERS IN THE
                                        STRING IF LENGTH.NE.0.
```

```
BASIC MPU 8080/Z80      GATES/ALLEN/DAVIDOFF      MACRO %50(272)=1 21:12 21-FEB-78 PAGE 1-11
F3      MAC      17-FEB-78 19:18      SOME EXPLANATION

                                        MEANINGLESS OTHERWISE.
                        HIGH
                THE VALUE OF A STRING VARIABLE (THESE 4 BYTES)
                IS CALLED THE STRING DESCRIPTOR TO DISTINGUISH
                IT FROM THE ACTUAL STRING DATA. WHENEVER A
                STRING CONSTANT IS ENCOUNTERED IN A FORMULA OR AS
                PART OF AN INPUT STRING, OR AS PART OF DATA, STRLIT
                IS CALLED, CAUSING A DESCRIPTOR TO BE BUILT FOR
                THE STRING. WHEN ASSIGNMENT IS MADE TO A STRING POINTING INTO
                BUF THE VALUE IS COPIED INTO STRING SPACE SINCE BUF
                IS ALWAYS CHANGING. "STRCPY" IS USED TO COPY                )
                STRINGS.

                STRING FUNCTIONS AND THE ONE STRING OPERATOR "+"
                ALWAYS RETURN THEIR VALUES IN STRING SPACE.
                ASSIGNING A STRING A CONSTANT VALUE IN A PROGRAM
                THROUGH A "READ" OR ASSIGNMENT STATEMENT
                WILL NOT USE ANY STRING SPACE SINCE
                THE STRING DESCRIPTOR WILL POINT INTO THE
                PROGRAM ITSELF. IN GENERAL, COPYING IS DONE
                WHEN A STRING VALUE IS IN BUF, OR IT IS IN STRING
                SPACE AND THERE IS AN ACTIVE POINTER TO IT.
                THUS F$=G$ WILL CAUSE COPYING IF G$ HAS ITS
                STRING DATA IN STRING SPACE. F$=CHR$(7)
                WILL USE ONE BYTE OF STRING SPACE TO STORE THE
                NEW ONE CHARACTER STRING CREATED BY "CHR$", BUT
                THE ASSIGNMENT ITSELF WILL CAUSE NO COPYING SINCE
                THE ONLY POINTER AT THE NEW STRING IS A
                TEMPORARY DESCRIPTOR CREATED BY FRMEVL WHICH WILL
                GO AWAY AS SOON AS THE ASSIGNMENT IS DONE.
                IT IS THE NATURE OF GARBAGE COLLECTION THAT
                DISALLOWS HAVING TWO STRING DESCRIPTIONS POINT TO THE SAME
                AREA IN STRING SPACE. STRING FUNCTIONS AND OPERATORS
                MUST PROCEED AS FOLLOWS:
                        1) FIGURE OUT THE LENGTH OF THEIR RESULT
                        2) CALL GETSPA TO FIND SPACE FOR THEIR
                        RESULT. THE ARGUMENTS TO THE FUNCTION
                        OR OPERATOR MAY CHANGE SINCE GARBAGE COLLECTION
                        MAY BE INVOKED. THE ONLY THING THAT CAN
                        BE SAVED DURING THE CALL TO GETSPA IS A POINTER
                        TO THE DESCRIPTORS OF THE ARGUMENTS.
                        3) CONSTRUCT THE RESULT DESCRIPTOR IN DSCTMP.
                        GETSPA RETURNS THE LOCATION OF THE AVAILABLE
                        SPACE.
                        4) CREATE THE NEW VALUE BY COPYING PARTS
                        OF THE ARGUMENTS OR WHATEVER.
                        5) FREE UP THE ARGUMENTS BY CALLING FRETMP.
                        6) JUMP TO PUTNEW TO GET THE DESCRIPTOR IN
                        DSCTMP TRANSFERRED INTO A NEW STRING TEMPORARY.

                THE REASON FOR STRING TEMPORARIES IS THAT GARBAGE
                COLLECTION HAS TO KNOW ABOUT ALL ACTIVE STRING DESCRIPTORS
                SO IT KNOWS WHAT IS AND ISN'T IN USE. STRING TEMPORARIES ARE
                USED TO STORE THE DESCRIPTORS OF STRING EXPRESSIONS.
```

```
BASIC MPU 8080/Z80     GATES/ALLEN/DAVIDOFF     MACRO %53(272)=1 21:12 21-FEB-78 PAGE 1-12
F3      MAC      17-FEB-78 19:16              SOME EXPLANATION

                         INSTEAD OF HAVING AN ACTUAL VALUE STORED IN THE
                         FAC, AND HAVING THE VALUE OF A TEMPORARY RESULT
                         BEING SAVED ON THE STACK, AS HAPPENS WITH NUMERIC
                         VARIABLES, STRINGS HAVE THE POINTER TO A STRING DESCRIPTOR
                         STORED IN THE FAC, AND IT IS THIS POINTER
                         THAT GETS SAVED ON THE STACK BY FORMULA EVALUATION.
                         STRING FUNCTIONS CANNOT FREE THEIR ARGUMENTS UP RIGHT
                         AWAY SINCE GETSPA MAY FORCE
                         GARBAGE COLLECTION AND THE ARGUMENT STRINGS
                         MAY BE OVER-WRITTEN SINCE GARBAGE COLLECTION
                         WILL NOT BE ABLE TO FIND AN ACTIVE POINTER TO
                         THEM. FUNCTION AND OPERATOR RESULTS ARE BUILT IN
                         DSCTMP SINCE STRING TEMPORARIES ARE ALLOCATED
                         (PUTNEW) AND DEALLOCATED (FRETMP) IN A FIFO ORDERING
                         (I.E. A STACK) SO THE NEW TEMPORARY CANNOT
                         BE SET UP UNTIL THE OLD ONE(S) ARE FREED. TRYING
                         TO BUILD A RESULT IN A TEMPORARY AFTER
                         FREEING UP THE ARGUMENT TEMPORARIES COULD RESULT
                         IN ONE OF THE ARGUMENT TEMPORARIES BEING OVERWRITTEN
                         TOO SOON BY THE NEW RESULT.

                         STRING SPACE IS ALLOCATED AT THE VERY TOP
                         OF MEMORY. MEMSIZ POINTS BEYOND THE LAST LOCATION OF
                         STRING SPACE. STRING ARE STORED IN HIGH LOCATIONS
                         FIRST. WHENEVER STRING SPACE IS ALLOCATED (GETSPA)
                         FRETOP, WHICH IS INITIALIZED TO [MEMSIZ], IS UPDATED
                         TO GIVE THE HIGHEST LOCATION IN STRING SPACE
                         THAT IS NOT IN USE. THE RESULT IS THAT
                         FRETOP GETS SMALLER AND SMALLER, UNTIL SOME
                         ALLOCATION WOULD MAKE [FRETOP] LESS THAN OR EQUAL TO
                         [STKTOP]. THIS MEANS STRING SPACE HAS RUN INTO THE
                         STACK AND THAT GARBAGE COLLECTION MUST BE CALLED.

                         GARBAGE COLLECTION:
                                 0. MINPTR=[STKTOP] [FRETOP]=[MEMSIZ]
                                 1. REMMIN=0
                                 2. FOR EACH STRING DESCRIPTOR
                                 (TEMPORARIES, SIMPLE STRINGS, STRING ARRAYS)
                                 IF THE STRING IS NOT NULL AND ITS POINTER IS
                                  .GT.MINPTR AND .LT.FRETOP,
                                 MINPTR=THIS STRING DESCRIPTOR'S POINTER
                                 REMMIN=POINTER AT THIS STRING DESCRIPTOR
                                 END
                                 3. IF REMMIN.NE.0 (WE FOUND AN UNCOLLECTED STRING)
                                 BLOCK TRANSFER THE STRING DATA POINTED
                                 TO IN THE STRING DESCRIPTOR POINTED TO BY REMMIN
                                 SO THAT THE LAST BYTE OF STRING DATA IS AT
                                 [FRETOP]. UPDATE FRETOP SO THAT IT
                                 POINTS TO THE LOCATION JUST BELOW THE ONE
                                 THE STRING DATA WAS MOVED INTO. UPDATE
                                 THE POINTER IN THE DESCRIPTOR SO IT POINTS
                                 TO THE NEW LOCATION OF THE STRING DATA.
                                 GO TO STEP 1.
                         AFTER CALLING GARBAGE COLLECTION GETSPA AGAIN CHECKS

BASIC MPU 8080/Z80     GATES/ALLEN/DAVIDOFF     MACRO %53(272)=1 21:12 21-FEB-78 PAGE 1-13
F3      MAC      17-FEB-78 19:16              SOME EXPLANATION

                         TO SEE IF [A] CHARACTERS ARE AVAILABLE BETWEEN
                         [STKTOP] AND [FRETOP] , IF NOT AN "OUT OF STRING"
                         ERROR IS INVOKED.

                 MATH PACKAGE
                         THE MATH PACKAGE CONTAINS FLOATING INPUT (FIN),
                         FLOATING OUTPUT (FOUT) FLOATING COMPARE (FCOMP)
                              AND ALL THE NUMERIC OPERATORS AND FUNCTIONS.
                         THE FORMATS, CONVENTIONS AND ENTRY POINTS ARE ALL
                         DESCRIBED IN THE MATH PACKAGE ITSELF.

                 INIT -- THE INITIALIZATION ROUTINE (FOR NON-T/S VERSION)
                         INITIALIZATION FIRST LOOKS AT THE SWITCH REGISTER
                         TO SEE WHAT TYPE OF I/O SHOULD BE DONE.
                         ANY NON-STANDARD I/O CAUSES LOCATIONS IN BASIC
                         TO BE CHANGED. THEN THE AMOUNT OF MEMORY,
                         TERMINAL WIDTH, AND WHICH FUNCTIONS TO BE RETAINED
                         ARE ASCERTAINED FROM THE USER. A ZERO IS PUT DOWN
                         AT THE FIRST LOCATION NOT USED BY THE MATH-PACKAGE
                         AND TXTTAB IS SET UP TO POINT AT THE NEXT LOCATION.
                         THIS DETERMINES WHERE PROGRAM STORAGE WILL START. THE
                         HIGHEST MEMORY LOCATION MINUS THE AMOUNT OF DEFAULTED
                         STRING SPACE (50) GIVES THE FIRST LOCATION USED BY THE
                         STACK. SPECIAL CHECKS ARE MADE TO MAKE SURE
                         ALL QUESTIONS IN INIT ARE ANSWERED REASONABLY, SINCE
                         ONCE INIT FINISHES THE LOCATIONS IT USES ARE
                         USED FOR PROGRAM STORAGE. THE LAST THING INIT DOES IS
                         CHANGE LOCATION ONE TO BE A JUMP TO READY INSTEAD
                         OF INIT. ONCE THIS IS DONE THERE IS NO WAY TO RESTART
                         INIT.
                 (IN THE TIMESHARING VERSION, THE INITIALIZATION
                  PROCESS IS COMPLETELY DIFFERENT. SEE TSINIT.MAC)

                 STORAGE
                         [TXTTAB]        A ZERO
                                         POINTER TO NEXT LINE'S POINTER
                                         LINE # OF THIS LINE (2 BYTES)
                                         CHARACTERS ON THIS LINE
                                         ZERO
                                         POINTER AT NEXT LINE'S POINTER
                                                  (POINTED TO BY THE ABOVE POINTER)
                                         ... REPEATS ...
                         LAST LINE:      POINTER AT ZERO POINTER
                                         LINE # OF THIS LINE
                                         CHARACTERS ON THIS LINE
                                         ZERO
                                         DOUBLE ZERO (POINTED TO BY THE ABOVE POINTER)
                         [VARTAB]        SIMPLE VARIABLES. 6 BYTES PER VALUE.
                                         2 BYTES GIVE THE NAME, 4 BYTES THE VALUE
                                         ... REPEATS ...
                         [ARYTAB]        ARRAY VARIABLES. 2 BYTES NAME, 2 BYTE
                                         LENGTH, VALUE (EXTRA IF LENGTH ON)
                                         ... REPEATS ...
                         [STREND]        FREE SPACE
                                         ... REPEATS ...
```



Storage layout for BASIC

(low memory)

	zero	(1 byte)
[TXTTAB]	pointer to next line	(2 bytes)
	binary line #	(2 bytes)
	character on line	(see note 1)
	zero	(1 byte)
	<Repeat above for each line>	
	zero	(2 bytes)
[VARTAB]	Simple variables. 6 bytes per variable 2 bytes give the name 4 bytes give the value. <Repeat for each variable>	
[ARYTAB]	Array variables 2 byte name. 2 byte length. values — Repeats for each array	
[STREND]	lowest location for stack	
	Free space (SP can be in here)	
[STKTOP]	most recent stack entry stack	
[FRETOP]	bottom of stack / ~~topest~~ location for strings	
	free space	
[~~FRETOP~~]	Current string usage STRINGS	
[MEMSIZ]	highest machine location.	

This scheme allows for simple table management. Only collector is for strings which aren't in 4K BASIC.

盖茨手写的8080 BASIC内存配置原始说明。

COMPUTER NOTES/SEPTEMBER,1975

Software Notes

by Bill Gates

Though the most difficult and enjoyable part of writing a program is the design of data structures and program flow, it is also important to use the least number of instructions possible to perform each function in a program. For instance:

```
CALL SUB1      should be replaced by
RET

JMP SUB1       unless something fairly
               tricky is being done
```

with return addresses. The JMP is faster, takes one less byte, and uses no stack space. An instruction book on programming the 8008 ignores this simple fact!

JMPs should be avoided wherever possible. By rearranging code you can often avoid having an unconditional JMP by falling into the routine you were JMPing to.

The beginning programmer will use lots of SHLDs, LHLDs, STAs and LDAs when they are not necessary. The stack can be used to save temporary values in most cases. SHLDs, LHLDs, LDAs and STAs should only be used for values referenced in many different contexts within a program, i.e. an I/O parameter or the current line number.

A good technique for familiarizing yourself with the instruction set is to go out of your way to use every instruction at least once (except perhaps DAA). Go through the instruction set from time to time and look closely at the instructions you seem to use very rarely. With few exceptions (DAA, SPHL) all the instructions can be used to advantage, even in small programs. One of the most overlooked instructions is XTHL. When all the accumulators have values that must be saved and a value needs to be taken off the stack, XTHL is the only instruction that can be used.

```
Example:  ;Exchange [B,C] with [H,L]

PUSH B    ;put [B,C] on the stack

XTHL      ;[H,L] = top stack entry =
          [B,C]

          ;[H,L] goes on the stack

POP B     ;[B,C] = original [H,L]
```

Sometimes the simple way of doing things is the best. PUSH B/POP D may seem like a tricky way of setting [D,E] = [B,C], but the obvious sequence MOV D,B/MOV E,C is much faster.

Some tricks involve instruction sequences which at first sight seem meaningless. For instance: SUB A or XRA A. Subtracting A from itself or exclusive-oring A with itself are the only one-byte ways of setting A=0. MVI A,0 must still be used if the condition codes need to be preserved, but this is rare.

ADC A is equivalent to RAL, except it affects all the condition codes. SBB A sets A=0 if carry is off and A=377 if carry is on. The routine below uses this fact to convert A as a signed integer to a double byte signed integer in [H,L]:

```
MOV L,A   ;setup the low order
          ;now the sign must be
          ;"extended" by setting H=0
          ;if A=>0 and H=377 otherwise
RAL       ;Carry = 1 if A<0
          ;Carry = 0 if A=>0
SBB A     ;A=0 if old A was =>0
          ;A=377 if old A was <0
MOV H,A   ;setup the high order
```

The sequence: INR E
DCR E

doesn't modify any values, but it does set the condition codes (except carry) depending on what is in E. If E is being used as a flag to indicate, say, whether or not a decimal point has been seen, the zero flag is set up to do a conditional JMP.

The subject of good decimal print routines has been discussed extensively in the Altair Software Department this week. This routine is one of the four or five I wrote this week -- each with its own advantages and disadvantages. This one is fairly tricky, in that it takes a little bit of looking at to understand.

```
#1 ;
   ;Print the binary unsigned number
   ;in [H,L] in decimal, suppressing
   ;leading zeros
   ;
   ;24 bytes (25 if saves D,E)
   ;ON RETURN:
   ;A = last digit in ASCII
   ;B,D = 255 (all constants in
   ;decimal)
   ;C,E = last digit -10
   ;H,L = 0
   ;
   ;Uses up to 18 bytes of stack
   ;Total compute time up to 85
   ;milliseconds
   ;
   ;IDEA: calculate a digit, save it
   ;      on the stack, and call the
   ;      digit calculator to calcu-
   ;      late and print higher order
   ;      digits, pop the digit off
   ;      and print it.
   ;
```

-continued

下列文章刊登于*Computer Notes*，一份由戴维·布奈尔（David Bunnell）制作、面向MITS Altair计算机用户的简报。盖茨的部分手稿显示在简报对应段落的旁边。盖茨解释说："这些（例程）展示了若干不可思议的技巧——真正把东西压缩到极致。其中打印例程是我们所能想到的接收二进制数并打印位数的最短例程。"
读纸带的引导程序必须用电脑前方的开关逐一键入Altair。盖茨写的引导程序在第3页的第3栏。他解释说："这是最简短的引导程序，非常非常难懂。这个程序人们得看上15分钟，才能弄清楚它是怎么回事。"

Print the binary unsigned number in [H,L]
in decimal. suppress leading zeros!

dea for digit printer:
calculate a digit - save it on the stack and call the digit calculator to calculate and print higher order digits, pop the digit off the stack and print it.

Uses up to 18 bytes of stack.
up to 85 millesecondc of compute.



Software Notes

```
DECOUT:   LXI B, -1Ø        ;CALL here
GETDIG:   MOV D,B           ;[D,E] = -1
          MOV E,B           ;since B = 255
LOOPSB:   DAD B             ;Subtract 1Ø from [H,L] until [H,L] < 1Ø. Carry
                            ;won't be set by the last DAD when [H,L] < 1Ø.
          INX D             ;increment the count
          JC LOOPSB         ;loop subtracting
          PUSH H            ;[L] = current digit -1Ø
                            ;Save the current digit on the stack.  Change to
                            ;XTHL and add PUSH D at GETDIG to save [D,E].
          XCHG              ;[H,L] = old [H,L]/1Ø
          MOV A,H           ;Set zero flag if [H,L] = Ø
          ORA L
          CNZ GETDIG        ;If not zero, print the higher order digits and
                            ;then return here to print this digit.
          MVI A, "Ø" + 1Ø   ;A = constant to add to digit
          POP B             ;pop the digit into C
          ADD C             ;A = ASCII of digit
          JMP OUTCHR        ;Jump to the routine to print A and return.  If
                            ;OUTCHR is located next, the JMP can be eliminated.
```

```
Best digit printer
Short digit print out 24 bytes
;enter at Decout with number in [H,L]
DECOUT:   LXI B,-10        ;base count
GETDIG:   MOV D,B          ;set [D,E]=-1
          MOV E,B          ;since [B]=255
LOOPSB:   DAD B            ;subtract 10 from [H,L]
                           ;until [H,L]<0.
          INX D            ;increment the count.
          JC LOOPSB        ;no carry if [H,L]>10.
          PUSH H           ;[L]=current digit-10
          XCHG             ;[H,L]=old [H,L]/10
          MOV A,H          ;see if [illegible] [H,L]=0
          ORA L            ;
          CNZ GETDIG       ;if not recursively
                           ;print the higher order
                           ;digits [illegible]
          MVI A,"0"+10     ;constant to get char code
          POP B            ;get the digit -10
          ADD C            ;compute char code
          JMP OUTCHR       ;print [A] and return.
```

Parity is used as a check to detect errors in data transmission. Each data word is given an additional bit which is set to 1 if there are an odd number of 1's in the data and Ø otherwise. When the data is received the parity bit is checked to make sure it is set properly. Thus, if you are reading a 7-bit ASCII paper tape with the 8th bit used for parity, the parity of the entire 8 bits should be even.

The reason I first thought about a parity routine for the 8080 is that the parity condition code and all the instructions related to it (JPO, JPE, RPE, RPO, CPO, CPE) are seldom used. I wondered how difficult it would be to calculate parity if the parity flag were removed. A user-settable flag would be much more useful than the parity flag. BASIC uses the parity flag in only about eight places, and all of these are special tricks. Here is the smallest parity routine I've been able to write:

```
;Enter with number in A.  1Ø bytes.
;On exit, A=Ø and all the other reg-
;isters are preserved.
;Carry is set depending on A's
;parity.
;Enter at ODDPAR for carry on to
;mean odd parity.

ODDPAR:   ADD A          ;Move a bit of A into carry.

          RZ             ;If all bits added into carry, return.

          JNC ODDPAR     ;If no bit moved into carry, rotate more.

;enter at EVNPAR for carry on to
;mean even parity

EVNPAR:   ADI 200        ;Complement the parity of the remaining bits

          JMP ODDPAR     ;Rotate more.
```

```
Lets say the 8080 had no
parity bit - minimal routine to do it.
They didn't need parity!
Is there a smaller routine?
; Compute parity.
; Enter with number in [A]
; on exit [A]=0 and all others preserved
; Carry is set depending on [A]'s parity
; enter at oddpar for carry on to mean
; odd parity
ODDPAR:   ADD A          ;move a bit into carry
          RZ             ;if all bits added, return
          JNC ODDPAR     ;if no bit in carry, rotate more
; enter at EVNPAR for carry on to mean even parity
EVNPAR:   ADI 200        ;complement parity of remaining
          JMP ODDPAR     ;rotate more.
```



Software

By bill Gates

Condition Codes

There seems to be some confusion about the condition codes. These are the Boolean (true/false) flags that are set/reset depending on the results of certain instructions. They are:

Z = zero - result was 0
S = sign - the most significant bit (MSB) of the result
P = parity - the result has an even number of ones in it
C = carry - an arithmetic operation generated a carry out of the most significant bit (i.e. adding 200 to 212)
CY_1 = first digit carry - this is used only for BCD arithmetic and will be elaborated on next month.

It is the condition codes that determine whether conditional JMP's, CALL's and RET's will be executed (i.e. RZ, CPE, JP). JM, CM, and RM (minus) are executed if the sign flag is on. JP, CP, and RP (positive) are executed if the sign flag is off. JZ, JNZ, CNZ, RZ, RNZ (zero/no zero) depend on the zero flag just as JC, JNC, CC, CNC, RC, RNC (carry/no carry) depend on the carry flag. CPE, JPE, RPE (parity even) are executed if the parity flag is on and CPO, JPO, RPO are executed when it is off.

The condition codes do not always reflect the value in A since IN, LDA, LDAX, MOV and MVI can change A but do not affect the condition codes. Instructions like INR C, DCR L, CMP B, CPI 3, STC, CMC and DAD B affect the condition codes, but not A.

Affect carry only: STC, CMC, RAL, RAR, RLC, RRC and DAD.

Affect all but carry: INR, DCR.

Affect all: ADD, ADC, SUB, SBB, CMP, ANA, ORA, XRA, DAA and their immediate counterparts (i.e. ADI,CPI).

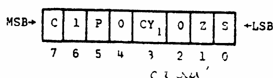
Use carry to affect result: CMC, RAP, RAL, ADC, SBB, ACI, SBI, DAA.

The instructions XRA, ORA, ANA, XRI ORI and ANI always reset carry.

If the condition codes do not reflect A's value (i.e. you just did a LDA or MOV into A) and you want to see if A=0, use ORA A or ANA A. CPI 0, ADI 0 and OPI 0 also work but they are 2-bytes.

The only other instructions besides the ones in the list above that use the condition codes are PUSH PSW and POP PSW. Respectively, they SAVE/RESTORE the condition codes and A on the stack.

For tricky programmers a sequence like PUSH B / POP PSW may be used to set the condition codes. This has the effect of moving B into A (MOV A, B) and moving C into the condition codes. The PSW (condition code) format is



Therefore if C was 201_8 before the POP PSW, zero and sign would be set and parity and zero would be unset. The bits marked '0' and '1' are constant and cannot be changed.

HINT #1

If you have a counter that can be bigger than 255 but is always less than 65535, it is convenient to use the following:

```
LXI B, count    ;set up counter
LOOP: code to be executed 'count'
      times
DCX B           ;decrement count
                ;does not affect
                ;condition codes
MOV A,B
ORA C           ;see if any bits set
JNZ LOOP        ;go back if so
```

HINT #2

For those who like to save bytes, and especially for those with 256-byte machines, (a byte is always 8 bits, which is a word on the 8800) RST's that are not used for interrupts, debug calls, monitor calls, etc. can be used to call subroutines that get called in many places (i.e. a character input subroutine). An RST is only 1 byte and a CALL is 3 bytes. Even if you have to put in a JMP so you don't overrun another RST location (0,10,20,30,40,50,60,70) you will probably save bytes.

Loading Software

Software from MITS will be provided in a checksummed format. There will be a bootstrap loader that you key in manually (less than 25 bytes). This will read a checksum loader (the 'bin' loader) which will be about 120 bytes.

For audio cassette loading the bootstrap and checksum loaders will be longer. All of this will be explained in detail in a cover package that will go out with all software.

For loading non-checksummed paper tapes here is a short program:

```
STKLOC: DW GETNEW
        (2 bytes-#1 low byte of
              GETNEW address
           #2 high byte of
              GETNEW address)

STAPT:  LXI H,0
GETNEW: LXI SP, STKLOC
        IN <flag-input channel>
        RAL  ;get input ready bit
        RNZ  ;ready?
        IN <data-input channel>
CHGLOC: CPI <043 = INX B>
        RNZ
        INR A
        STA CHGLOC
        PET
                     (22 bytes)
```

Punch a paper tape with leader, a 043 start byte, the byte to be stored at loc 0, the byte to be stored at 1, - - - etc. Start at START, making sure the memory the loader is in is unprotected. Make sure you don't wipe out the loader by loading on top of it.

To run this again change CHGLOC back to CPI - 376.





这两页摘自VisiCalc第一版手册。当时这个程序还叫CalcuLedger（CL）。注意布兰克林手写修改部分。

Using CL is similar to placing your pencil on a piece of paper, and then writing something down. The main ~~difference~~ advantage is that CL provides you with a "magic" piece of paper that has the ability to do computations like a calculator. Before we actually use CL, you ~~must~~ should learn the definition of a few terms that are used to refer to parts of the computer screen. Here is an example of a simple application using CL, as it would appear on the screen, with these parts labelled:

The screen is divided into two sections: the top three lines (the menu/status area) and the data area below. The data area is where data actually appears. The data area is divided up into rows and columns. The rows are numbered and the columns are named by letters. The intersection of a row and a column is called an entry. Each entry can be referred to by indicating its column and row, e.g. "B7". You can place either numbers (called values) or text (called labels) at an entry.



The highlight on entry B5 above is called the cursor. You can move the cursor from entry to entry using the arrow keys on the right of the keyboard. This is the first thing that you should learn to do in order to use CL. The character in the upper right corner of the screen (called the direction indicator) will tell you which way the arrow keys will make the cursor move. If the direction indicator is a minus sign (-) then the cursor will move left and right. If it is an exclamation point (!), then the cursor will move up and down. The right arrow always moves you forward (from A to B, or 1 to 2) while the left arrow always moves you backward (from B to A, or 2 to 1). [You can switch the direction indicator between ! and - by pressing the space bar.] Try moving the cursor around on the screen to get used to how it works. You might notice that the beginning

Replicate

The replicate command is used to copy the contents of an entry into other entries. It is useful when a group of entries (such as a row) are all similar expressions. For example, you could set an entry to be the sum of the two entries above it. You could then use the replicate command to make the other entries in that row also be the sum of the two entries above them. Entries whose contents have been set using the replicate command are no different than entries that have been set by hand, and can be displayed by pointing to them with the cursor and modified individually just like any other entry.

Any type of entry may be replicated -- expressions in value entries, label entries, blank entries~~, and the other special entry types~~. If the entry is a value entry, then the replicate command can make ~~a few~~ changes to the expression as it copies it into each new entry. ~~For example,~~ when a reference ~~to another entry~~ appears in an expression, the replicate command gives you ~~the~~ option ~~of having~~ the copy refer to the exact same entry (by not modifying the reference), ~~or~~ having it refer to the entry that is in the same relative position to the entry with the copy as the originally referenced entry was to the entry being copied. ~~You can also make modifications to numeric values that appear in an expression. You can have the value incremented by a constant amount each time that it is copied~~. The examples at the end of this section should help you understand how these expression modifications work.

To use the replicate command you first type /R. It will then ask you what you want to replicate. It will have the current entry already filled in on the input line (line 3 of the menu/status area). You ~~then position the cursor on the entry that you want to copy (if its the current entry then you don't have to do anything), and then~~ press RETURN. CL will then



THE COMPUTER UTILITY AS A MARKETPLACE FOR COMPUTER SERVICES

by

Robert Matthias Frankston

Abstract

Computers are unique in their ability to be programmed for a wide variety of applications. This is in contrast with hardware dedicated to specific tasks such as the telephone system. Because of its flexibility, a computer system can support, concurrently, many diverse services that do not require dedicated hardware. Conversely, these services act to bring the capabilities of the computer to the consumer who might otherwise find the operational difficulty of running computer programs too formidable.

Since the computer is supporting many services which are sold to consumers it is natural to model the system as a marketplace for these services. Most contemporary computer systems are oriented towards users who run programs. The environment for services puts different requirements on the computer systems than do the needs of programmers, so as to permit all the participants in the market to make effective use of its facilities without requiring dedicated facilities and without interfering with each other. As with any marketplace, it must be convenient to do business within its framework.

The requirements of such a marketplace are not satisfied in contemporary ccmputer systems. However, the marketplace can be evolved from some existing computer systems without fundamental changes. Presently the use of a ccmputer requires considerable expertise on the part of the user. The evolution to a marketplace is necessary if the capabilities of computer systems are to be made more widely available than they are now.

THESIS SUPERVISOR: F. J. Corbato
TITLE: Professor of Electrical Engineering



Thesis directions as of December 17, 1976 R. Frankston

Introduotion

I am interested in allowing the user of a personal (i.e. local) database system to make effective use of data residing in other such systems. This would be accomplished by providing the user with a means of specifying the characteristics of his database so that a datamanagement system may be able to take advantage of the these characteristics in improving performance.

I am assuming an environment consisting of many computer systems. Each system is built around a small, but sophisticated computer. It may be thought of as a personal Multics (for example) or even as a process on a shared system. The key concepts are:

1. There is no central authority that can dictate policy to these systems.

2. The topology of the connections between such systems varies. We are interested in solutions that are effective for a class of interactions rather than being optimized for one particular configuration. Closely related to this is the assumption that no system can "know" the overall topology.

3. The users of such systems will vary from sophisticated users who may be concerned with dealing with their systems as computers to more common users who do not want to have to specify how to perform each function in order to get useful work done.

Systems in such an environment would be oriented more toward information management than computation, or even "data processing". For the sake of simplicity I am going to equate the term "information management" with "data management". The former term is actually more appropriate but encompasses intelligent processing that I prefer not to deal with at this time. For the sake of my model I am going to assume the availability of a database technology including query languages that can deal with a local database. My interest is then in extending such a technology to allow a user of each system to deal with data that may reside at other such systems. By this I mean that the datamanagement systems should provide an interface that allows a user to share information (again, just data for now) with users of other systems without being concerned with the details of how the sharing is done. It must do this while providing a level of efficiency and consistency with his conceptual model of this process that is comparable with that obtainable within the context of a local datamanagement system.

Perhaps this examples does not belong here. While it is detailed, it is perhaps too much so this early in a proposal.

As an example of an environment with distributed information management we can examine three database sites within a corporation, concentrating on management of salary and payroll data. The three participants are a branch office, the personnel department and the payroll department. This introduction gives a brief example of a hypothetical interaction which we will examine in more detail later.

DRAFT 1 December 17, 1976 0:10



鲍勃·弗兰克斯顿提供了一页硕士论文和一页博士论文，阐明他在1970年代中期的一些想法。他说："即使现在也一样。尽管这个模型已经不再是分时服务的组成部分，但大方向仍是个人能够利用可用的信息。"

```
/*
Cubic spline fitting - Ellis-McLain Method
Jonathan Sachs
22-Oct-85

This method generates a single-valued function from an ordered set of x,y data.  It
produces the best results when used on smooth functions since it reduces discontinuities in
the second derivative.

The findgrad procedure is called first to determine slopes at each data point.  This step
may be omitted if slopes are already given for each data point.  Next the coeff procedure
is called for each interval to generate the coefficients of the cubic polynomial that fits
the data in that interval.

Reference:

Ellis, T.M.R. and McLain, D.H. (1977) "Algorithm 514 - A new method of cubic curve fitting
using local data".  ACM Transactions on Mathematical Software, Volume 3, pages 175-178.
*/

/*      find gradient at each data point

        findgrad(x,y,grad,n)

        x               n-vector of x coordinates of data points
                        values must be in increasing order with no two equal

        y               n-vector of y coordinates of data points

        grad    returned n-vector

        n               number of data points (must be at least 4)

*/

findgrad(x,y,grad,n)
double x[];
double y[];
double grad[];
int n;
{
        int i,iless2,iless1,iplus1,iplus2;
        double x0,x1,x2,x3,x4,y2;
        double prod1,prod2,num,denom,g;
        double coeff2,xdiff,xprod,weight;

        for(i = 0; i < n; i++) {

                iless1 = i > 0 ? i-1 : i+3;
                iplus1 = i < n-1 ? i+1 : i-3;

                x2 = x[i];
                y2 = y[i];

                x1 = x[iless1] - x2;
                x3 = x[iplus1] - x2;

                prod1 = x3*(y[iless1]-y2);
                prod2 = x1*(y[iplus1]-y2);

                denom = x1*x3*(x[iless1]-x[iplus1]);

                g = (x1*prod2-x3*prod1)/denom;
                coeff2 = (prod1-prod2)/denom;

                if(i <= 1) {
                        num = denom = 0.0;
                }
                else {
                        iless2 = i-2;
                        x0 = x[iless2] - x2;
                        xdiff = x[iless2] - x[iless1];
```

乔纳森·萨奇用这段程序说明了他用C语言（一种专为优化运行时间而设计的高级编程语言）编写代码的风格。



```
			xprod = x0*xdiff*(x[iless2]-x[iplus1]);
			weight = xprod/(xdiff*xdiff);
			num = weight*(y[iless2]-y2-x0*(g+x0*coeff2));
			denom = weight*xprod;
		}

		if(i <= n-3) {
			iplus2 = i+2;
			x4 = x[iplus2] - x2;
			xdiff = x[iplus2] - x[iplus1];
			xprod = x4*xdiff*(x[iplus2]-x[iless1]);
			weight = xprod/(xdiff*xdiff);
			num += weight*(y[iplus2]-y2-x4*(g+x4*coeff2));
			denom += weight*xprod;
		}

		grad[i] = g + num*x1*x3/denom;
	}
}

/*	calculate the coefficients of the cubic which is used for the interval
	from x[i] to x[i+1]

	coeff(x,y,grad,i,&c0,&c1,&c2,&c3)

	x			x coordinate vector

	y			y coordinate vector

	grad		gradient vector (see above)

	i			index specifying interval to be fitted

	fitted cubic has equation:

	c0 + c1*(x-x[i]) + c2*(x-x[i])^2 + c3*(x-x[i])^3

*/

coeffs(x,y,grad,i,c0,c1,c2,c3)
double x[];
double y[];
double grad[];
int i;
double *c0;
double *c1;
double *c2;
double *c3;
{
	double dx;
	double dx2;
	double dy;
	double g1;
	double g2;

	dx = x[i+1] - x[i];
	dx2 = dx*dx;
	dy = y[i+1] - y[i];
	g1 = grad[i];
	g2 = grad[i+1];

	*c0 = y[i];
	*c1 = g1;
	*c2 = (3.0*dy-dx*(2.0*g1+g2))/dx2;
	*c3 = (dx*(g1+g2)-2.0*dy)/(dx*dx2);
}
```



下面这8页笔记是罗伯特·卡尔写给自己看的，意在理清字处理/电子表格程序Framework开发中的早期思路。

3-15-82 ①

FRAME MANAGER

Good morning Robert. Your task is to figure out
1) frame organization in the db. How are they organized, accessed, threaded.
2) proc (frame) accessing & threading

- All frames have a universal 32-bit address
  - on 8088 this yields 16-b. seg addr & 16-b logical addr
- all pointers to other frames consist of a 32-bit name. Passing this frame name through a translation table gives a 32-bit frame pointer (above mentioned)

At runtime we have the frame space - a heap or string space like memory area. All frame refs are via the frame name. As this passes through the translation/lookup code, if the frame isn't in the space, it's brought from disk. If it's not on disk, an error occurs.

At one end of frame space is the frame stack. The frame mngr also includes the core interpreter for the frame language. The interpreter & language use the stack. When a cell is evaluated it's first put on the stack and all its referenced cells & frames are evaluated.

② 3-15-82

will the Frame Language be Reverse Polish Notation? (Prefix)

This is how values are kept up to date. Simple grind it through EVAL of whole reference tree.
-or do we want a clever scheme of knowing what's been changed & what hasn't?

now, where do values live? and what does it mean for a frame to be sent to the execute port?

- each cell has at least 2 corresponding cells (a 3rd for format?). 1 for the formula "frame language instance" or text; 1 for the "value" of that f.l. instance. If all the cell has is text, as in a paper, then it doesn't make sense for it to have a value. If it does have a f.l.i. then the value is empty until the first time the user sends the f.l.i. to the execute port.

No. the "value cell" corresponding to the formula cell will be in a totally different frame, a temporary 'twin' frame cloned from the first only for the life of the value port.

Also, probably any kind of formatting info, if we have it, should go to a separate frame.

The execute port keeps the source & value frames associated.

Ex.



③

Frames then are a collection of variable length cells.
Cells can have at least 3 types of data:

- text
- text that is a frame language instance
- number

Q: cell typing? What will we have? if cell types support sequences and records & arrays then it'll be a cinch to implement all indices and info about frames ontop of the frames

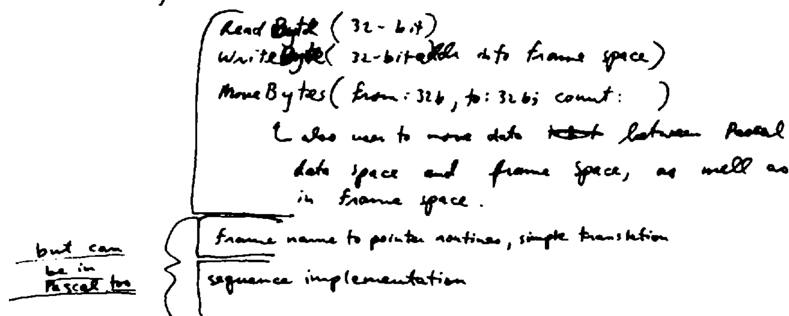
Frames have some kind of frame map, to tell where cells are, & what they are too.
I want frame cells to be easily changeable in size, deletable, insertable.

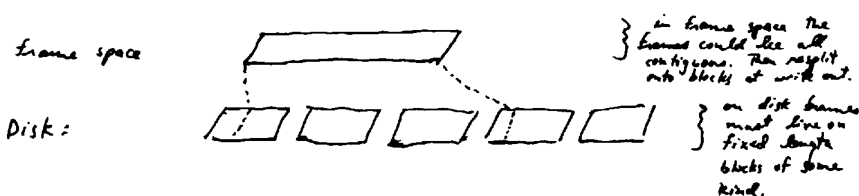
If a frame cell of plain text is a sequence, then the editor obviously has an easy time of it.

Since the frame space will be in a different part of memory than Pascal, there'll have to be read & write word procs, then most all the frame mngr can be written in Pascal. Then, later, what needs to be faster can be written into assembler & deal with that memory directly. Then the pascal code will be additional documentation for the key assemble routines.

(4)

initially in assembler we'll need





Ok, for now we'll have the plan:

- frames are (potentially) split across blocks on disk.
- but when a frame is read in (possibly forcing other frames out!) it is read into a contiguous chunk.
- finally the frame is split across blks again at write out time.

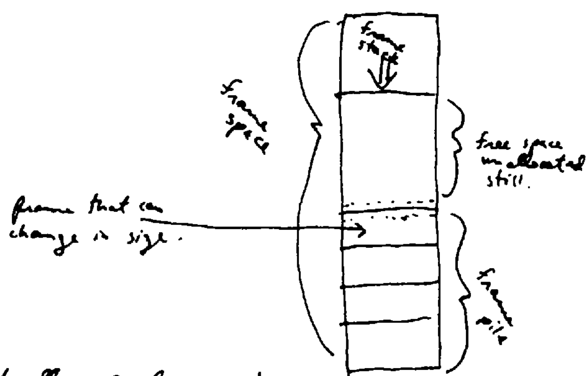
I choose this design ↑ because I envision memory being big enough for a frame space large enuf to hold most all frames. Hence at system startup we read all frames in, at system turn off we write them all out.

But what about a frame changing its size, it'll force all the others to shift in memory?

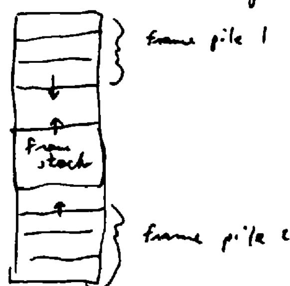
⟶ No, the current frame, or any frame that's about to



be changed in size, will first be brought to the top of the frame pile, next to free space. Since all editing is confined to one frame at a time (except multiple windows? and couldn't value frames also change size at execute time if we allow string types for cells & values?) only the current frame to change will be changing in size. And we've just moved him to the top of the frame pile (and shifted over his old spot).



if we wanted to allow 2 frames to concurrently change, we'd have frame space like so:

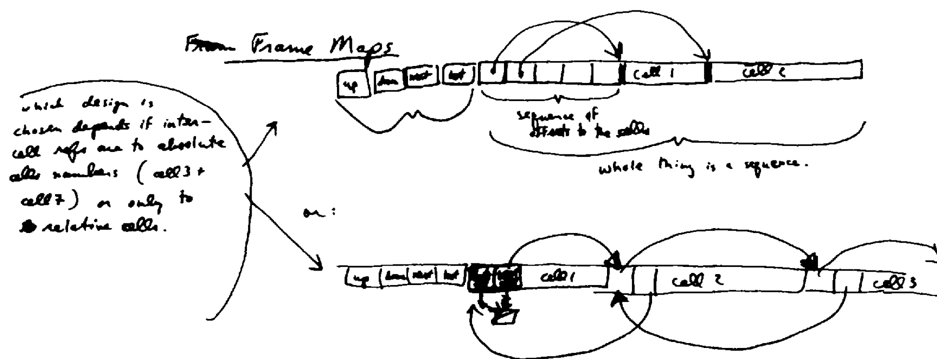


or some such monstrosity.

⑥

But I like the first picture of frame space. It'll work well: 1) multiple frames that are often changing in size "concurrently" will be small ones, easily copied to & from the top of the pile. The changing ones will collect in a working set at the top of the pile so not much shifting has to be done

2) when the user's edits 2 or more large frames, almost always only one will be edited and the others only read or copied. – is this really true? I hope so. I envision the frames most always being small, anyways.



the typing
bootstrapping

⑦

Frame Linking

these procs in assembler.

Frame name to pointer translation table (a table of which frames are in mem and where) ~~[crossed out]~~
a sequence of 2 32-bit entries.

Then otherwise, whatever kinds of linking I decide for.

e.g.:

new Framename := Up, Down, Next, Last (Frame name);

⑧

Frame Cells

how to give frame cells meaningful names for reference?

have absolute or relative (or both) inter-cell refs?



⑨

Current Position = Current Curson Loc

globally there is one current pos. This consists of:

- cur Port ?
- cur Frame
- cur Cell
- cur byte in Cell.

one for every Port

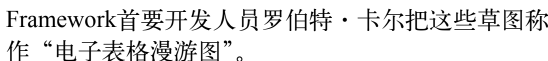
Now how is a cell displayed? sent to it's bitmap which is then sent to the graphics buffer?

So what happens when one character is inserted into a cell?

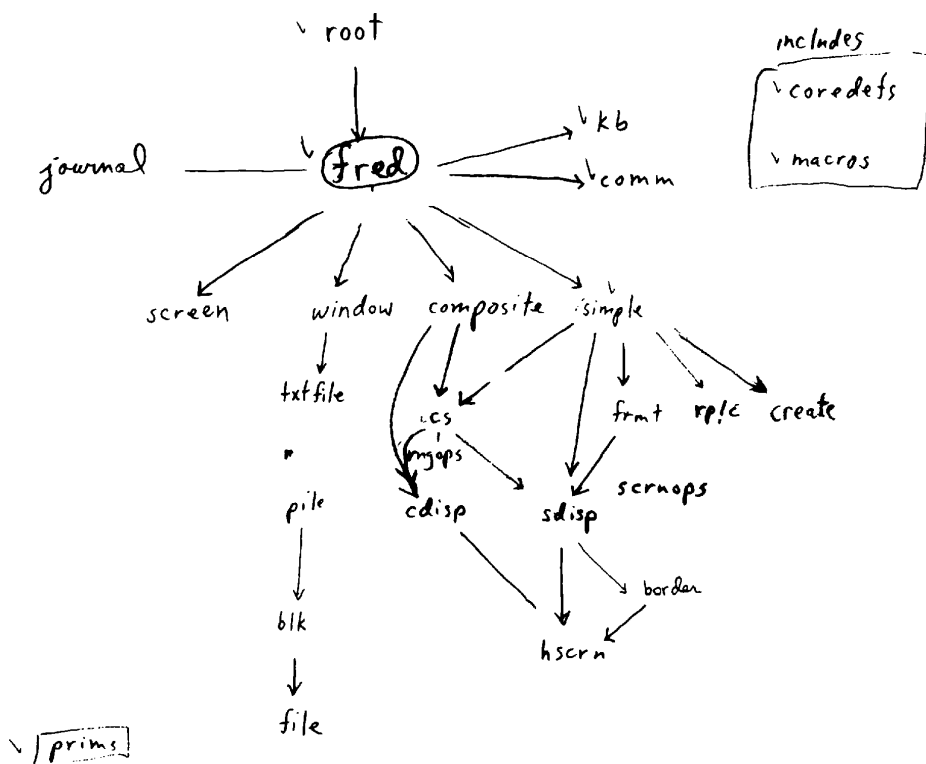
1) the char is inserted into the cell itself, adjusting pointers, after 1st move the cells frame to the top of the frame pile if necessary

2) that cell is then redisplayed to the bit map
3) which bitmap is then redisplayed to the graf buffer

2 redisplays of whole cell sounds like alot for every character change to my mind, but it gives up the whole dynamicness of the display etc: dynamically changing justification in cells etc., as you type into a cell it gets bigger & bigger, other cells get pushed down, etc. etc.
And all with relatively simple code. As soon as I try to only redisplay the current line only, then the code gets much worse.



Framework首要开发人员罗伯特·卡尔把这些草图称作“电子表格漫游图”。



Register Usage

bx points to csd
di points within cs fr

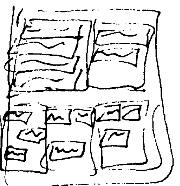


罗伯特·卡尔的这张示意图显示了“内部代码结构如何各就其位”。“Fred”是Framework的开发代号。



罗伯特·卡尔提供的Framework菜单树状结构图。

Menus:



Hierarchical Tree of menus

at lowest level are single frames:

each button is a choice

some choices may be on at same time

each choice specifies the other sibling choices which must be off, & those which must be on (eg. hitting [Justify] turns [Fill] button on also).

each choice has an associated action routine

User moves through menu w/ normal commands and does a [doit] on selection

each choice has a msg which is displayed when it's selected

all invoked choices redisplay immediately their action



Properties:

fr: left, cntr, justify } position props

formatting: fill, justify, no-fill } text props

normal (integer) dollars full detail (scientific) } number props

{there are no units, user can easily edit numbers to put those in)

italic, bold, underline, blink, inverse, uppercase, lowercase } character props

protected } ca

罗伯特·卡尔提到的在Framework开发中的“头脑风暴、排序、算法和结构”实例。

OLD: ONE COMPUTER – MANY PEOPLE

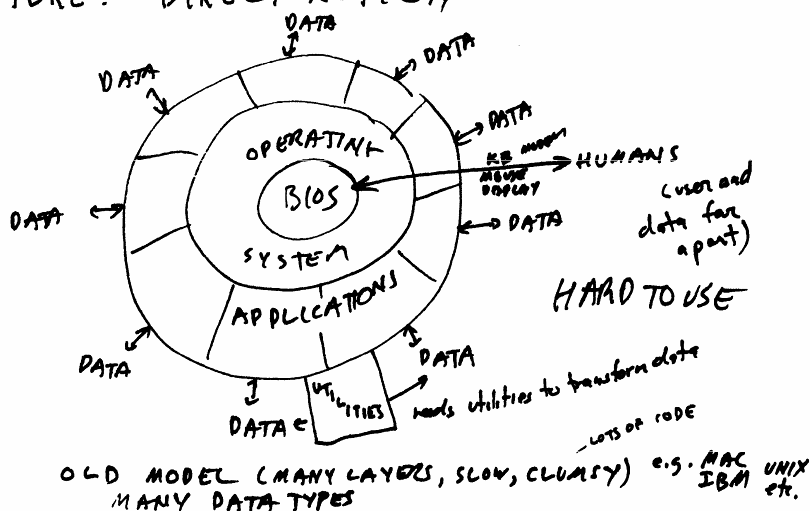
NOW: ONE COMPUTER – ONE PERSON

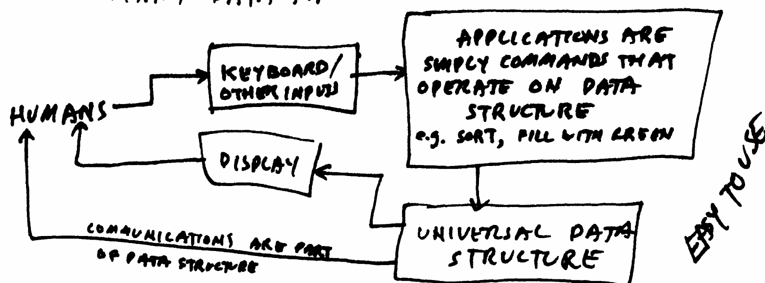
FUTURE: ONE PERSON – MANY COMPUTERS

OLD: COMMAND LANGUAGES

NOW: WINDOWS/ICONS/MENUS

FUTURE: DIRECT ACTION





NEW MODEL, NO O.S. FAST, SIMPLE e.g. SWYFT, CAT

These concepts led to starting Information Appliance Inc in 1982

Jef Raskin 1987
hand copied 1989

杰夫·拉斯金提供的草图，其中展示的那些想法，促使他在1982年创办了信息设备公司。



安迪·赫兹菲尔德，Macintosh操作系统首要开发人员，新近开发了Switcher程序和ThunderScan数字化仪，提供了这个Macintosh原始程序。他说："这个程序……会让图标在屏幕上快速跳动起来。"

```
; File IconBounce.TXT
;-----------------------------------------------------------------------
;
;   IconBounce uses custom plotting routines to bounce a lot of icons
;   around on the deskTop.
;
;     written by Andy Hertzfeld, Nov 17 1985
;
;-------------------------------------------------------------------------

INCLUDE MacSys:MacTraps.D

; System definitions, etc.

IOCompletion    EQU     12              ;offset to completion routine address
IOFileName      EQU     18              ;offset to fileName
IOVRefNum       EQU     22              ;offset to volume refNum
IOFileType      EQU     26              ;offset to type byte, permissions
IOMisc          EQU     28              ;offset to misc param
IOBuffer        EQU     32              ;offset to buffer pointer
IOByteCount     EQU     36              ;offset to count
IONumDone       EQU     40              ;offset to number done
IOPosMode       EQU     44              ;offset to positioning mode
IOPosOffset     EQU     46              ;offset to position value

EvtMsg          EQU     2               ;offset to message field
EvtMeta         EQU     14              ;offset to metaKey field
Where           EQU     10              ;mouse offset in event record
portRect        EQU     16              ;offset to portRect

ScreenRow       EQU     $106            ;rowBytes of screen [word]
Ticks           EQU     $16A
KeyMap          EQU     $174
Time            EQU     $20C
ScrnBase        EQU     $824
CurApRefNum     EQU     $900
WmgrPort        EQU     $9DE

MaxX            EQU      512
MaxY            EQU      342

; Icon Data Structure

NextIcon        EQU     0               ;handle of next structure
IconData        EQU     4               ;128 byte 32 by 32 bitmap
IconMask        EQU     132             ;128 byte mask
IconPosition    EQU     260             ;longword position
IconVelocity    EQU     264             ;velocity

IconDSSize      EQU     268             ;total data structure size

; Global Variable Definitions

QuickBase       EQU     -4              ;quickDraw globals
MyEvent         EQU     QuickBase-200   ;my event record
QuitFlag        EQU     myEvent-2       ;boolean for exiting
WhichWindow     EQU     QuitFlag-4      ;whichWindow result

NumIcons        EQU     WhichWindow-2   ;# of icons allocates
FirstIcon       EQU     numIcons-4      ;handle of 1st one
LastIcon        EQU     FirstIcon-4     ;handle of last one

BigBuffer       EQU     LastIcon-4      ;pointer to big buffer


XDEF            START

START:

; first allocate some zeroed space by clearing it off the stack

                MOVE    #511,D0         ;need about 2K bytes
ClearLoop
```



```
                CLR.L   -(SP)
                DBRA    D0,ClearLoop

; now grow the heapZone as large as we can make it

                MOVEQ   #64,D0
                SWAP    D0                  ;get huge number
                _NewHandle                  ;grow out the heap
                BNE.S   InitWorld           ;we expect the error
                _DisposHandle               ;if it not, dispose it

; initialize QuickDraw and the toolBox

InitWorld
                PEA     QuickBase(A5)       ;push address of QuickDraw vars
                _InitGraf                   ;initialize QuickDraw
                _InitFonts                  ;initialize the font manager
                _InitCursor                 ;get the arrow cursor
                _InitWindows                ;initialize the window manager
                _InitMenus                  ;ditto for menus

                CLR.L   -(SP)               ;our recovery proc is  NIL
                _InitDialogs                ;initialize dialogs
                _TEInit                     ;and text edit, too

                BSR     SetHourGlass

; initialize our globals

                CLR.B   QuitFlag(A5)
                CLR.W   numIcons(A5)

                CLR.L   FirstIcon(A5)
                CLR.L   LastIcon(A5)

; allocate the big buffer

                MOVE.L  #24000,D0
                _NewPtr
                BNE     ErrorExit
                MOVE.L  A0,BigBuffer(A5)

; display title message

                MOVE.L  WmgrPort,-(SP)
                _SetPort
                PEA     BigRect
                _ClipRect

                MOVE.L  #$000E0038,-(SP)
                _MoveTo

                PEA     TitleString
                _DrawString

; allocate the icons

                BSR     AllocIcons

                _HideCursor

                MOVEQ   #31,D0
                _FlushEvents

; start the main event loop

MainLoop
                _SystemTask

                BSR     HandleEvent         ;check for events and handle them
                BSR     AnimateIcons

                TST.B   QuitFlag(A5)
```



```
                BEQ.S   MainLoop

                _ExitToShell            ;back to finderLand

; HandleEvent checks for events and handles them as necessary.  It handles
; the menu commands and all interaction with the user.

HandleEvent
                SUBQ    #2,SP           ;make room for result
                MOVE.W  #-1,-(SP)       ;we want every event
                PEA     myEvent(A5)     ;stick it in our global
                _GetNextEvent           ;get the event

                TST.B   (SP)+           ;did we get one?
                BEQ.S   NoEvent         ;if not, we're done

                MOVE.W  myEvent(A5),D0  ;get the event number
                BEQ.S   NoEvent         ;ignore the Null event
                CMP     #9,D0           ;only care about 1st 9 events
                BGE.S   NoEvent

                ADD     D0,D0           ;double for word index
                LEA     EvtDispatch,A0  ;get the address of the table
                ADD.W   0(A0,D0),A0     ;get routine address
                JMP     (A0)            ;go to it!

; here is the event dispatch table

EvtDispatch
                DC.W    NoEvent-EvtDispatch
                DC.W    MyMouseDown-EvtDispatch
                DC.W    MyMouseUp-EvtDispatch
                DC.W    MyKeyDown-EvtDispatch
                DC.W    NoEvent-EvtDispatch
                DC.W    MyKeyDown-EvtDispatch
                DC.W    MyUpdateEvt-EvtDispatch
                DC.W    MyDiskInsert-EvtDispatch
                DC.W    MyActivate-EvtDispatch
MyDiskInsert
MyMouseUp
MyActivate
NoEvent
                RTS


; Handle keyboard events

MyKeyDown
                ;ST     QuitFlag(A5)
                RTS

; handle update events

MyUpdateEvt
                RTS

; the following routine handles mouseDowns.  First call FindWindow
; to classify where the mouse went down

MyMouseDown

                ST      QuitFlag(A5)
                RTS

; AllocIcons opens the desktop, and allocates an icon data structure for
; each ICN# in the file.

AllocIcons
                SUBQ    #2,SP
                PEA     DeskTopName
                _OpenResFile
                TST     (SP)
                BMI     ErrorExit
```



```
                SUBQ     #2,SP
                MOVE.L   ICNRType,-(SP)            ;push ICN# type
                _CountResources                    ;get # of resources
                MOVE.W   (SP)+,D3                  ;keep in D3
                BLE      ErrorExit

; limit the # of icons to 64, unless the option key is down

                BTST     #2,KeyMap+7
                BNE.S    NoILimit

                CMP      #64,D3
                BLE.S    NoILimit

                MOVEQ    #64,D3

; OK, now loop for each icon

NoILimit
                MOVEQ    #1,D4                     ;init index
AllocIcLoop
                SUBQ     #4,SP                     ;make room for result
                MOVE.L   ICNRType,-(SP)            ;push ICN#
                MOVE.W   D4,-(SP)                  ;push index
                _GetIndResource                    ;get it
                MOVE.L   (SP)+,D5                  ;got it?
                BLE      DoneAllocIcon

; we have the icon handle, so allocate the structure

                MOVE.L   #IconDSSize,D0
                _NewHandle
                BNE      ErrorExit

                MOVE.L   A0,A3                     ;get new handle
                MOVE.L   (A3),A2                   ;handle->ptr

                CLR.L    (A2)+                     ;link is zero
                MOVE.L   A2,A1                     ;set up dest

                MOVE.L   D5,A0
                MOVE.L   (A0),A0                   ;set up source
                MOVE.L   #256,D0                   ;256 bytes to move
                _BlockMove

                ADD.L    #256,A2                   ;skip over save area

; generate positions 0 < x < 512,  0 < y < 302

                SUBQ     #2,SP
                _Random
                MOVEQ    #0,D0
                MOVE.W   (SP)+,D0
                DIVU     #302,D0
                SWAP     D0
                ADDQ     #1,D0
                MOVE     D0,(A2)+

                SUBQ     #2,SP
                _Random
                MOVE.W   (SP)+,D0
                AND.W    #511,D0
                ADDQ     #1,D0
                MOVE     D0,(A2)+

; generate velocitys from 1 to 8

                SUBQ     #2,SP
                _Random
                MOVE.W   (SP)+,D0
                AND      #7,D0
                ADDQ     #1,D0
```



```
                MOVE.W   D0,(A2)+

                SUBQ     #2,SP
                _Random
                MOVE.W   (SP)+,D0
                AND      #7,D0
                ADDQ     #1,D0
@2
                MOVE.W   D0,(A2)+

; link it in the list

                MOVE.L   LastIcon(A5),D0
                BNE.S    LinkItIn

                MOVE.L   A3,FirstIcon(A5)
                MOVE.L   A3,LastIcon(A5)
                BRA.S    BumpICount
LinkItIn
                MOVE.L   A3,LastIcon(A5)
                MOVE.L   D0,A0
                MOVE.L   (A0),A0
                MOVE.L   A3,(A0)                 ;link it in
BumpICount
                ADDQ     #1,numIcons(A5)

DoNextIcon
                ADDQ     #1,D4                   ;bump index
                CMP      D3,D4                   ;done?
                BLT      AllocIcLoop
DoneAllocIcon
                _CloseResFile

                RTS

; ShowIcon is the routine that plots an icon.  It is adopted from the
; BigCursor routines.  The handle of the icon data structure is passed
; in A3.

ShowIcon
        MOVEM.L D0-D7/A0-A4,-(SP)        ; save registers
        MOVE.L  (A3),A3                  ; de-reference icon data structure

        MOVEQ   #32,D5                   ; size of icon
        MOVEQ   #16,D6                   ; half size

        LEA     IconData(A3),A2          ; cursor data bitmap address
        LEA     IconMask(A3),A4          ; cursor mask bitmap address

;  first handle the x coordinate

        MOVE    IconPosition+2(A3),D0    ; get left
        MOVEQ   #15,D2                   ; upper left X-coordinate
        AND.W   D0,D2                    ; bit offset within word

        AND     #$FFF0,D0                ; truncate to nearest word
        BGE.S   @0                       ; if positive, skip

        MOVEQ   #0,D0                    ; minimum upper left X-coord of 0
        ADD.W   D6,D2                    ; adjust right shift count

; if shift count > 15, just move over a word

@0
        CMP     D6,D2                    ;is it?
        BLT.S   @7                       ;if not, skip

        SUB     D6,D2                    ; reduce bit index
        ADD.W   D6,D0                    ; bump base point
@7

; establish "last word" boolean
```



```
        CLR.W   -(SP)

        MOVE    D0,D1                   ; copy coordinate
        SUB.W   #MaxX-32,D1             ; upper left X-coord <= 512-32
        BLT.S   @2                      ; branch if <= 512-32

        MOVE.W  #MaxX-32,D0             ; maximum X-coord of 512-32
        ADD.W   D1,D2                   ; adjust left shift count
        ST      (SP)                    ; set the boolean

; handle the y coordinate

@2      MOVE.W  D5,D4                   ; 32 rows

        MOVE.W  IconPosition(A3),D1     ; get Y-coordinate

;   Display the icon on the screen.

        MOVE.L  BigBuffer(A5),A1        ; offscreen memory address
        LSR.W   #3,D0                   ; convert X-coord to bytes
        ADD.W   D0,A1                   ;  and add to screen address
        MOVE.W  ScreenRow,D5            ; bytes per row on screen
        MULU    D5,D1                   ;  * Y-coord
        ADD.L   D1,A1                   ;  added to screen address

        SUBQ    #4,D5                   ; bias D5 for loop

        TST     D2                      ; is shiftcount = 0?
        BEQ.S   BotFastLoop             ; if so, go ultra fast

; OK, for added speed, two different loops, depending on the
; if we need the 3rd word (as specified by the top of stack boolean)

        TST.B   (SP)+
        BNE     BotCurlLoop
        BRA.S   BotCurLoop              ; test for rows=0

; here is the icon plotting loop.  First do the leftmost 32 bits

ShowCurLoop
        MOVE.L  (A2)+,D0                ; get the data
        MOVE.L  D0,D6                   ; copy for later
        LSR.L   D2,D0                   ; shift into place

        MOVE.L  (A4)+,D1                ; get the mask
        MOVE.L  D1,D7                   ; copy for later
        LSR.L   D2,D1                   ; shift into place
        NOT.L   D1                      ; complement mask

        AND.L   D1,(A1)                 ; bit-clear with the mask
        OR.L    D0,(A1)+                ; plot the data

; now handle the rightmost 16 bits

        MOVEQ   #16,D1
        SUB.W   D2,D1                   ; compute left shift count

        ASL.W   D1,D6                   ; shift the data
        ASL.W   D1,D7                   ; shift the mask
        NOT.W   D7                      ; complement the mask

        AND.W   D7,(A1)                 ; bit clear the mask
        OR.W    D6,(A1)                 ; plot the data
@0
        ADD     D5,A1                   ; bump to next row
BotCurLoop
        DBRA    D4,ShowCurLoop          ; loop till done

DoneShowLoop
        MOVEM.L (SP)+,D0-D7/A0-A4       ; restore registers
DoneShow
        RTS
```



```
; this loop is used when we're near the right edge and don't have to
; plot the third word

ShowCurlLoop
        MOVE.L  (A2)+,D0                ; get the data
        LSR.L   D2,D0                   ; shift into place

        MOVE.L  (A4)+,D1                ; get the mask
        LSR.L   D2,D1                   ; shift into place
        NOT.L   D1                      ; complement mask

        AND.L   D1,(A1)                 ; bit-clear with the mask
        OR.L    D0,(A1)+                ; plot the data

        ADD     D5,A1                   ; bump to next row
BotCurlLoop
        DBRA    D4,ShowCurlLoop         ; loop till done

        BRA     DoneShowLoop

; special fast loop for the 6% case where we don't have to shift

FastLoop
        MOVE.L  (A2)+,D0                ;fetch the data
        MOVE.L  (A4)+,D1                ;fetch the mask
        NOT.L   D1                      ;complement mask

        AND.L   D1,(A1)                 ;plot the mask
        OR.L    D0,(A1)+                ;plot the data

        ADD     D5,A1
BotFastLoop
        DBRA    D4,FastLoop

        ADDQ    #2,SP                   ;discard boolean
        BRA     DoneShowLoop

; Error handling routines

ErrorExit
        DC.W    $F123

        RTS


; AnimateIcons is the mainline routine that bounces the icons

AnimateIcons
        BSR     GrayBuffer              ;fill big buffer with gray

        BSR     UpdateIconPositions
        BSR     DrawIntoBuffer

; now move the buffer onto the screen with blockMove

        MOVE.W  ScreenRow,D0
        MULU    #20,D0
        ADD.L   ScrnBase,D0
        MOVE.L  D0,A1                   ;screen is destination
        MOVE.L  BigBuffer(A5),A0        ;big buffer is source

        MOVE.L  #20608,D0
        _BlockMove

        RTS

; DrawIntoBuffer goes through the icon data structure, drawing each icon

DrawIntoBuffer
        MOVE.L  A3,-(SP)                ;save work reg

        MOVE.L  FirstIcon(A5),D0        ;get first one
        BEQ.S   DoneDIB                 ;if empty, skip
```



```
DIBLoop
        MOVE.L  D0,A3

        BSR     ShowIcon

        MOVE.L  (A3),A0
        MOVE.L  (A0),D0
        BNE.S   DIBLoop
DoneDIB
        MOVE.L  (SP)+,A3
        RTS

; UpdateIconPositions animates the icon positions

UpdateIconPositions

        MOVE.L  FirstIcon(A5),D0        ;get first one
        BEQ     DoneUIP                 ;if empty, skip

UIPLoop
        MOVE.L  D0,A0
        MOVE.L  (A0),A0

        MOVE.L  IconPosition(A0),D0
        MOVE.L  IconVelocity(A0),D1

; OK, bounce in the x position

        ADD.W   D1,D0                   ;compute new position
        BGE.S   @0                      ;if > 0, skip

        SUB.W   D1,D0                   ;undo it
        NEG.W   D1                      ;toggle velocity
@0
        CMP.W   #510,D0
        BLT.S   BounceY

        SUB.W   D1,D0
        NEG.W   D1

BounceY
        SWAP    D0
        SWAP    D1

        ADD.W   D1,D0
        BGE.S   @0

        SUB.W   D1,D0
        NEG.W   D1
@0
        CMP     #302,D0
        BLT.S   NextBounce

        SUB.W   D1,D0
        NEG.W   D1

NextBounce
        SWAP    D0
        SWAP    D1
        MOVE.L  D0,IconPosition(A0)
        MOVE.L  D1,IconVelocity(A0)

        MOVE.L  (A0),D0
        BNE     UIPLoop
DoneUIP
        RTS


; GrayBuffer fills the 322 scan lines at GrayBuffer with gray

GrayBuffer
```



```
        MOVE    #160,D2                 ;# of scan line pairs - 1
        MOVE.L  BigBuffer(A5),A0        ;point to the buffer
FillGLoop
        MOVE.L  #$55555555,D0           ;get gray
FillGLoop2
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long
        MOVE.L  D0,(A0)+                ;fill a long

; OK, now a scan line is done, so flip the gray

        NOT.L   D0
        BMI.S   FillGLoop2

        DBRA    D2,FillGLoop2

        RTS

; SetHourGlass installs the hourGlass (watch) cursor

SetHourGlass

        SUBQ    #4,SP
        MOVE    #4,-(SP)
        _GetCursor

        MOVE.L  (SP)+,A0
        MOVE.L  (A0),-(SP)
        _SetCursor

        RTS


; Constants, etc.

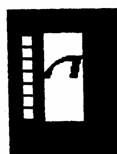
ICNRType
        DC.B    'ICN#'
DeskTopName
        DC.B    7,'DeskTop'
BigRect
        DC.W    0,0,1000,1000
TitleString
        DC.B    60,'IconBounce by Andy Hertzfeld...  Press mouse button '
        DC.B    'to exit. '
```





斯科特·金的这些草图将数学运算展示为随机的、不精确的符号，其创建过程像绘图一样。

这些草图展示了小写字母a的可能演变过程。



E/S

Sat Sep 27, 1986
9:30 PM

Ascending S looks best. Too bad width is irregular. Preserves open bowl counter in E.

Too tight. E looks like S.

O bowl (for A/O, etc.)

How do we fit an O between two vertical lines? Obviously has to be narrower This is not tilted enough.

This is too pinched upper left outside, up rt in

This is a better shape, quite harmonious with O.

Narrow V (e.g. Stravinsky)

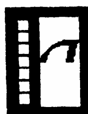
Position V in cell so T can be added.

Again, wider spacing wins. even tho this means unusual char width.

Narrow Y (for Y/L)

Narrow W

Reshape V so two adjacent make W, just as with ordinary V.





斯科特·金在1986年用MacPaint针对点阵字型做的研究，刊登于*Inversions for the Macintosh*（现名*Letterforms and Illusion*，W.H. Freeman出版）。

Hyper text for C programmers.
This one turns music upside down.
E-Z!

```c
#include <stdio.h>
#include <midi.h>
#d transform(f) ███████(f,stdout,a,b,A,B)

FILE *
Open(s){
        FILE *f = sopen(s,"r");
        return f? f : OpenTune(s);
}

pitch(s) char *s; { return (*s >= '0' && *s <= '9')? a

main(ac,av) char *av[]; {
        Int i=1, a=dx7_MIN, b=dx7_MAX, A=dx7_MAX, B=dx
        FILE *f;

        setbuf(stdout,NULL);
        for_each_argument {
        Case 'r': a = pitch(argument); b = pitch(argum
                  A = pitch(argument); B = pitch(argum
        Case 'R': a = pitch(argument); b = pitch(argum
                  B = a; A = b;
        Default: MidiError("%s [-r a b A B] [-R a b] [files or stdin]\n", av0);
                 MidiError("e.g., '%s -R 60 72' means 'reflect notes in the middle octave\n",av0);
                 exit(1);
        }

        if (i==ac) transform(stdin);
        else while (i<ac){
                if (f = Open(av[i],"r")) transform(f), sclose(f);
                else perror(av[i]);
                ++i;
        }
        exit(0);
}
```

```c
███████(in,out,low,h
        FILE *in, *out;
/*
** Copy the midi data from
** 'low...high' to 'Low...High' (using 'TransformMpuCmd()'.
** Example:
** .Cs
** MidiTransform(in,out,dx7_MIN,dx7_MAX,dx7_MAX,dx7_MIN)
** .Ce
** inverts pitches on the dx7 keyboard.
*/
{
        MpuCmd m;
        while (GetMpuCmd(in,&m)) {
                TransformMpuCmd(&m,low,high,Low,High);
                PutMpuCmd(out,&m);
        }
}
```

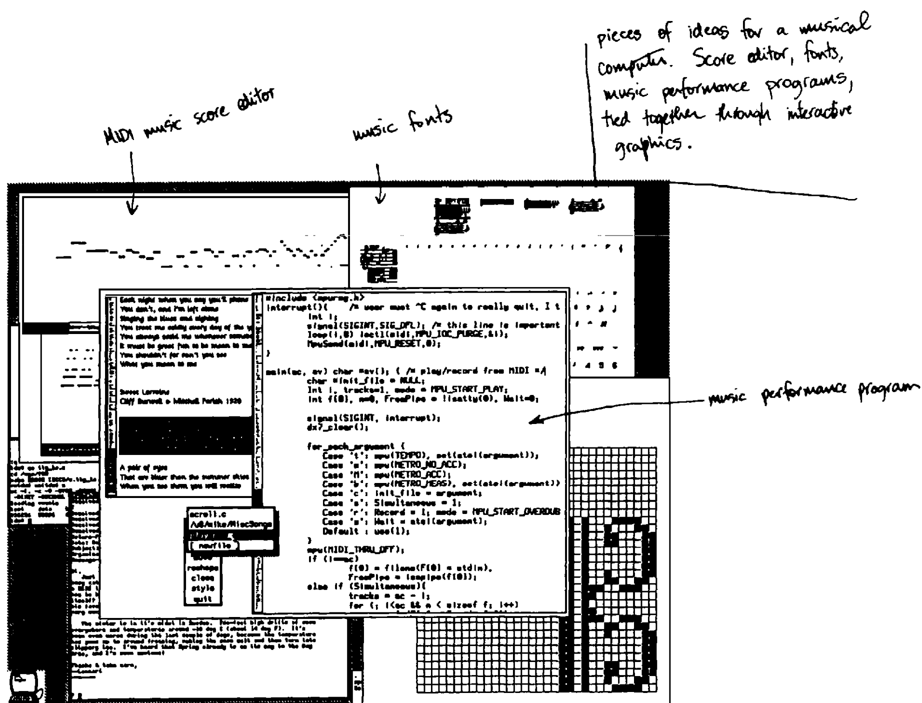
```c
#define md(a,b,c) ((long)(a)*(long)(b))/(long)(c)
#define transform(v, a, b, A, B) md(B-A, v-a, b-a) + A

███████(m, a, b, A, B)
        MpuCmd *m;
        unsigned char a, b, A, B;
/*
** Transform the pitch of 'MpuCmd' 'm' by scaling it from
** the range 'a...b' to the range 'A...B'.
** Pitches outside the range 'a...b' are unchanged.
** For instance, if pitches are in the range '30...50',
** the call 'TransformMpuCmd(m, 30, 50, 50, 30)'
** inverts the pitches.
*/
{
        register unsigned char p;
        if (IsNote(m) && (p=MpuPitch(m)) >= a && p <= b)
                MpuPitch(m) = transform(p, a, b, A, B);
}
```

迈克尔·霍利的程序会反转所有音调，将乐谱上下颠倒过来演奏。



来自卢卡斯影业的迈克尔·霍利分享了他对音乐计算机的一些想法。他手写的3页说明描述了屏幕的不同组件和支持语言。



graphical patching of microcodes
- starts as a picture; then ① becomes a language
implentation follows.

[Revbus] [Sbus]

[Lbus][Cbus][Rbus]

mike

jon's box

Delete: left button to delete item, mi
delete region.

Wait! uh...

c
c.c
cmd.c
font
list.h
md
slice.bits
slice.pic
symbols
tex
x.c
x2pan2
x3pan1

xab1
xadc1
xadd
xdac1
xeq1
xeq2
xfader1
xrevsend1
xslice.bits
xslider
xxfade1
xxfade2

② graphical patch language scratchings ...

n
w e
s

type
name
icon

eq ["eq-%" eq-icon

plugs ~~in {Vn}~~

[Vn] in {<e} cfreq boost {Vs} out

v, ^, >, <

size

no shadow, shadow

no box, box

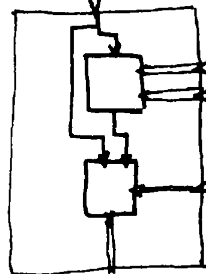




] at

join ["name"]

eq [...] → eq [...] → eq [...]



eq patch [~~[illegible]~~ size
eq (x, y)
xf (x, y)
~~join 2~~
~~eq/out → xf/in2~~
plugs {Vn} in {eq/in, xf/in1}
<e freq = eq/cfreq boost = eq/boost toggle = ~~xf~~/toggle
~~size~~
eq/out → xf/in2
]

Programmers at Work

Interviews With 19 Programmers Who Shaped the Computer Industry

编程大师访谈录

"Susan Lammers为我们成功呈现了一本独具风格的图书，它是大胆而有价值的'创意世界'丛书中的一本：对19位明星程序员进行和善而有深意的采访，让受访者自己畅谈……强烈推荐这本书，无论是对计算机迷，还是对厌恶计算机的人，这都是一本极有趣的读物。"

——《计算机周刊》

"《编程大师访谈录》之于初出茅庐的程序员，正如《巴黎评论》之于未来的小说家，不仅可以给人们带来安慰和启发，还能直观地感受到行业顶尖人物的思想……这本书绝对不容错过！"

——《纽约时报》

《编程大师访谈录》由一系列面对面的访谈组成，带领读者一起去探索计算机行业极具威望的19位编程大师的思想。这些访谈突出了塑造并影响当今软件业的先驱的推动力、事件和人物。这些编程大师们如何走上软件业的道路？他们爱用什么风格进行程序设计？编程是一种天赋还是一项可习得的技术？是艺术还是科学？编程大师们对于计算的未来有什么前瞻性的观点？本书为我们一一道来。

每篇访谈都可看作独特的成功故事，19篇合在一起就生动地刻画出上世纪80年代中期整个PC产业蓬勃兴起的一幅异彩纷呈的画卷。你可以从中看到莲花软件公司、苹果电脑公司、施乐帕洛阿尔托研究中心和微软公司等著名软件公司的草创和发展演变过程。

受访者包括安迪·赫兹菲尔德（苹果Macintosh操作系统开发者）、约翰·沃诺克（PostScript语言开发者）、C. 韦恩·莱特莱夫（dBASE的作者）、乔纳森·萨奇（Lotus 1-2-3的联合作者）和比尔·盖茨（BASIC语言开发者）等为软件业发展做出了卓越贡献的编程大师。附录形象地展示了这些软件奇才们的一些实际代码片段和工作底稿。我们还特意汇编了有关这些人物后来生活经历的内容。

- 19位业界先驱的访谈实录
- 领悟编程大师的深邃思想和深刻洞见
- 走出自己精彩的编程人生

ISBN 978-7-115-26431-2



ISBN 978-7-115-26431-2

定价：59.00元

图灵社区：www.ituring.com.cn

反馈/投稿/推荐信箱：contact@turingbook.com

热线：(010)51095186转604

分类建议 计算机/IT人文

人民邮电出版社网址：www.ptpress.com.cn